PATENT

METHOD AND APPARATUS FOR DISPLAYING DATA STORED IN LINKED NODES

This application claims the benefit of U.S. Provisional Application No. 60/135,740, filed on May 25, 1999.

FIELD OF THE INVENTION

The invention is a computer implemented method of storing, manipulating, accessing, and displaying data and its relationships, and a computer system (with memory containing data) programmed to implement such method.

BACKGROUND OF THE INVENTION

Today's GUI (graphical user interface) software requires a great deal of training for proficiency. They rarely show their context and rarely show internal data and control points. The actions of applications should be fast and easy to use, not confusing and restrictive.

Shortcomings of current technologies include the following:

Awkward Navigation: Navigating a current GUI is to learn how the programmers have organized the data and the control points. Today's UI (user interface) relies on windows and pull-down menus which hide prior ones as the user makes selections, and an often-complex series of mouse selections to navigate to the desired control point. In performing an action after traversing a path of pull-down menus and button selections on different sub-windows, the user may be returned to the beginning, forced to retrace their steps if they wish to apply an action repeatedly. The UI does not shift from one state to another in a smooth way.

Relationships are Invisible: Current GUI's with pop-up menus and layers of text-based windows. Today's GUI's suffer from a list mentality: rows of icons with little or no meaning in their position, no relationships readily visible between themselves or to concepts which the user may be interested in or have knowledge of.

The standard GUI concept of a canvas with buttons and pull-downs require reading and thinking about the meaning of text to, hopefully, grasp the model of control flow. Virtually no information is given outside of static text to help the user understand relationships and control points.

Access to Crucial Data is Confusing: Gaining access to fields or parameters in current applications can also be a challenge. For example, the Microsoft POP mail retrieval service uses a configuration profile defined by the user to access the mail server, and displays the name of that configuration when logging. Although one sees evidence of the existence of this configuration, how does one change it? The answer may not be obvious. An object-oriented system that implemented all actions on an object could also provide the mechanism of self-deletion, but this helps only if the object itself is visible and accessible via the GUI. This is what DataSea does. Windows technology is moving in this direction, by having many words in menus modifiable, but DataSea achieves this automatically by virtue of its design.

GUI Design is bug-prone: A complex GUI, such as a presentation and control system for database administration, today consists of many canvases and widgets which hopefully present, through multiple mouse clicks, all information that is available. Any data can be changed by any part of the program, and this leads to bugs if the programmer can not easily see these interactions. A DataSea presentation of data and control shows all the objects and their relationships, and thus shows immediately what nodes can affect changes to the data, reducing bugs. To turn a DataSea view into an "application" means to set internal parameters, create and link together application nodes, and add programmatic instructions to these nodes. DataSea will implement a means to invoke these instructions.

Interoperability Conflicts: DataSea can serve as the single source of data for any application. Any RDBMS (relational database management system) can do this, but DataSea is completely flexible in its data storage and linkage, guaranteeing forward compatibility by eliminating the risk of changes to database structure and entity-relationships of RDBs (Relational Databases).

Two separate DataSea databases can be joined, and automatic linkage routines will merge them without programmer effort. This is generally impossible in RDBs. This joining can occur by simply adding nodes and links from the two data sets, and adding together the contents of the master index, NameListObj. Or, the two data sets can be blended, merging their contents: taking two nodes with the same name from the two separate data sets, creating one node which has all the links from the two separate nodes.

In most storage systems, especially RDBMS's, the user must know how information is stored in the computer and which parameter or parameters that the computer is using to store the data, as well as the proper range of values. This is often non-intuitive and may seem somewhat arbitrary for a non-technical user. Ideally, the computer would better mimic human associative memory, allowing the user to look for new information associated with that which is better known, such as a particular context, or a range of values without regard to parameterization, to specify the target of interest.

OLAP (online analytical processing) and data mining require both analytical models and custom code to apply these models to particular database structures. These customizations may be hard-coded, non-portable and irrelevant to the model. The DataSea API (application programming interface) provides access to all data while eliminating the need to worry about database structure such as tables, columns and foreign keys.

Shortcomings of more recent data presentation technologies include the following:

Non-linear Viewing: Up-coming non-linear presentation tools such as fish-eye or hyperbolic views do not address the difficult problem of how to lay out the data and their relationships before the viewing method is applied. These may be useful, but do not address the difficult issue of how the graph is laid out initially. Nor are they appropriate for highly linked data sets, because the plethora of links resembles a cobweb from a psychotic spider.

Virtual Reality: VR takes advantage of visual clues and spatial awareness, but only for data sets that may be appropriately mapped to a 3-dimensional space. Generally data is N-dimensional and thus, in general, virtual-reality which models information as physical objects in 3D space is inappropriate for viewing arbitrary data.

Voice Interface: even more than a GUI, voice control needs a smooth transition from state to state in response to commands so that the user can follow what is happening. GUIs hide previous states with new windows, while the present invention moves objects gradually and continuously in response to programmatic or user events.

SUMMARY OF THE INVENTION

The inventive method (referred to as "DataSea") is a method for storing, accessing, and visualizing information. Data is stored into nodes and visualized as a 'sea' of linked nodes. Nodes can contain anything such as documents, movies, telephone numbers, applications or words containing concepts. Interactions with the user organize the data from a defined, and then refined, point of view, with relevant data brought to their attention by smooth changes in the data's appearance.

Essentially, a handful of nodes (which are typically selected by value) are linked to the point of view turning the web of data into a hierarchical network. Further order is imposed by the use of two types of commands: one which relies on the data values, the other on the links and types of the data nodes.

The user typically enters words into a computer programmed in accordance with DataSea and watches DataSea's response. Individual nodes are rendered according to the sequence of nodes between themselves and the point of view, allowing different presentations of data. Applications may be stored into nodes and can be located and executed by DataSea. These applications can operate on and present information from DataSea in the conventional manner, with windows and menus, or by using the DataSea mechanisms of visualization, including the so-called virtual reality mode ("VR-mode") which supports deterministic data placement as needed in such things as forms and spread sheets.

Examples of use include:

1. Entering keywords and phrases selectively retrieves and emphasizes different data types such as loose notes and email, but these entered keywords and phrases need not match exactly the content of the resulting emphasized data.
2. Direct and specific information is retrieved: The user enters the name of "Jim Smith" followed by an appropriate command such as "back" or "and" along with the phrase "Phone number" which refers to a pre-existing AN. Jim's telephone number becomes obvious as it approaches the point of view, which in this case is "Jim Smith".
3. A manufacturing facility has test data from machines and 3-D models of those machines. These data sources are integrated and the user can visualize the facility from different points of view, e.g. a virtual reality mode, tabular presentation or the standard DataSea network

connectivity display.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a diagram of relationships between nodes which can be visualized in accordance with the invention.

Fig. 2 is a flow chart of steps performed in accordance with the invention.

Fig. 3 is a block diagram of an embodiment of the invention.

Each of Figs. 4, 5, 6, 7, 8, and 9 is an example of a display generated in accordance with the invention.

PREFERRED EMBODIMENT OF THE INVENTION

This is a disclosure of computer implemented methods for storing, manipulating, accessing and displaying data and its relationships, and of computer systems programmed to implement such methods.

These methods are flexible, applicable to any kind of data and are useful to diverse groups of users. A prototype is implemented in the Java programming language.

Data is stored into nodes which are linked together. All nodes contain variables, including descriptions, types, magnitudes and timestamps. Links also contain information about themselves, including connection strength of the link and descriptive information.

Data is accessed and modified based on the values of data, their relationships, the values of DataSea parameters and links between nodes, rather than pre-determined locations in memory, as is done in most programming models.

Any existing application can be emulated in DataSea by creating and linking appropriate nodes. Positions of nodes as displayed on the screen are a result of processing force parameters rather than pre-determined positions.

Commands to DataSea are not chosen from hierarchical lists or menus, but rather they are

1. Simple navigation commands;
2. Words (data values) generated by the user; and
3. References to nodes brought to user's attention.

This approach to the command interface, and the smooth changes of state in visual feedback, lends DataSea to voice input and thus wireless `PDA' (personal digital assistant) type devices.

Key aspects of the invention are:

1. nodes which can contain any type of data, links between the nodes;
2. manipulation of internal parameters of nodes and links;
3. visualization of nodes and their internal parameters;
4. smooth transitions during changes in state of the network;
5. integration of virtual reality representations;
6. simple commands and low learning curve;
7. increased robustness to imprecise commands as the data set grows and matures;
8. obviation of the need to predetermine data structures when entering new data; and
9. integration of legacy data with the structure inherent in it.

DEFINITIONS:

- `POV' stands for `Point of View' and is the designation given a node from which many operations are begun, such as defining a hierarchy of distance in terms of links of any node to the POV.
- `DN' refers to a data node containing specific values of a parameter. `AN' refers to an abstract node, which contains a summary, abstract, or explanation of the data stored in one or more DNs, and represents a concept or parameter name which can link and thereby group one or more DNs. E.g., "address" is an AN, while the specific address "123 Main St." is a DN.
- DN("abc") refers to a data node named "abc". AN("xyz") refers to a AN named "xyz".
- 'magnify' means to change the value of the magnitude variable ('mag') or related variables inside a node and or its links.
- 'distance' refers to the minimum number of links between two nodes, also called the link-distance.
- 'conceptual distance' refers to a more complex function of link distance and other parameters including mag, link-connection-strength and total number of links to a node. For instance, the conceptual distance between two nodes which are several links apart can be proportional to the product of the link connection-strengths times the product of the mag's divided by a function of the link-distance.
- 'near' refers to a relatively low distance or

conceptual distance.

- 'commands' in DataSea are any programmatic methods of accessing, manipulating, creating or deleting data structures or elements of DataSea.
- 'applications' in DataSea are programs which use or modify data, links or resources of DataSea, and modify internal parameters of nodes or links, and or create or delete nodes.
- 'environment checking' refers to getting information about nodes and links in the neighborhood of or nearby one or more nodes.
- 'abbreviated' refers to incompletely rendering and positioning nodes.
- 'distal path' is a sequence of linked nodes, each node distal to the prior one.
- 'multiple paths' refers to more than one route, through links, between two nodes.
- 'interaction with DataSea' refers to either user interaction or programmatic interaction. User interaction is typically through the use of DataSea query language (keyboard or voice interface) and mechanical means such as a mouse. Programmatic interaction can result from DataSea nodes themselves or external programs.
- 'distal' refers to those nodes having a higher link-distance value (node.dist) than the link-distance value (this.dist) of the reference point or 'this': that is, node.dist > this.dist.

  Proximal is the opposite, referring to lower distance.
- 'primary node' refers to a node directly connected to the reference point or 'this'.

- 'child' is a primary distal node, while 'parent' is a primary proximal node.
- 'setting a POV' means assigning an existing node as a point of view, or creating a new node and linking it directly to a number of existing nodes, specified at the time of creation of the POV or later.
- "spreading mode" refers to the rules used in applying algorithms, possibly recursively, from one node to its neighbor or neighbors. Criteria might be dependent on proximal or distal progression, mag, CS (Connection Strength of a link), potentiation level, or other factors.
- 'Aliasing' refers to the mapping of a node to other nodes or a range of nodes. For instance, when an action is performed on the "tomorrow" node, it executes enough code to link itself to a node having the (absolute) value of tomorrow's date, dropping links to any other dates previously established. Linking a node to "today" results in linking to the node representing the absolute date. Depending on the spreading mode, related times will be more or less affected by commands. Nodes with TimeStamps closer to tomorrow will be affected more than those further away. Spreading can also be dependent on values and a range, e.g. decreasing the effect of "more" as 1/((current_value-optimal_value)/fuzziness_range). Integration of a GPS receiver, or simply assigning a static location in a map, will be used to alias the word "here".

"Presented data set" is what the user sees at the moment, being a collection of nodes with (mag > some threshold).

"Target data set" is an ideal collection of nodes representing the information the user is searching for.

"Traveling distally" means going from node A to node B only if B.dist is > A.dist.

"Traveling proximally" means going from node A to node B only if B.dist is < A.dist.

"Traveling downstream" means going from node A to node B only if A.getPol(B) is '-'. The polarization of the link between A and B can be set on entering the link between A and B, the order of A and B determining the polarization:

A.link(B, {polarization=}true)

means

A.getPol(B) is '-'. Typically, nodes A and B are linked such that node B is downstream from node A when node A is more general than node B (so that by traveling downstream from node A to Node B, a user encounters more specific, rather than less specific, information). In characterizing a node, a user usually wants to see progressively more general (broader) information about it.

"Traveling upstream" means going from node A to node B only if A.getPol(B) is '+'.

A *"Recently visited"* node denotes a node being used in an operation which traverses the network. The node's TimeStamp is updated on a visit.

As shown in Fig. 1, each node has a link to at least one other node. Each link is defined by three values: CS (which is Connection Strength of a link, initially set to 1.0), Description (which is a free-form String describing the node), and Type (For example DN (data node) and AN (abstract node)).

Fig. 2 is a flow chart of steps performed in accordance with the invention. First, a Point of View (POV) is set. Then, internal parameters associated with the POV are set. The internal parameters include those listed (with "pressure" denoting the values of summed forces influencing a node's position, "forces" denoting parameters indicative of pushing or pulling of the node's position relative to other nodes: positive or negative values calculated by repetitive interactions between any node and others, and global forces such as a `drift' or `gravity' biasing the positions of nodes, and "positions" denoting the location on the computer display, or more generally the doublet or triplet representing a node inside a virtual space, viewed on the computer's display). Then, feedback is provided to the user by displaying representations of the nodes on a display device in accordance with the internal parameters set in the second step. Optionally, auditory feedback is also provided to the user during the third step. As a fourth step, the user interacts by providing commands for modifying the internal parameters (both node parameters, such as magnitude, and link parameters, such as distance) set in the second step. The commands can include any of those listed, or any of those discussed below. In response to a command specified in the fourth step, the second step is performed again (to reset the internal parameters) and then the third step is performed again (to provide feedback indicative of the reset parameters).

EXAMPLE OF USING THE INVENTION:

A candy factory supervisor performs the following tasks (there are many different commands and choices of values which will give similar results):

- views the facility (the candy factory) to get an overview of its status and sees a VR (virtual reality) rendering of the various stations, color-coded for activity (by using the command "show factory");

- reviews the recent temperature history of one of the problem machines, a chocolate melter and sees the machine with clusters of data nearby

  ("reset, show Choc_2_Melter and Temperature and TimeLine and recent");

- views the melter and groups data from it and sees the abstract nodes "Temp", "Up-Time" and "Operators" which categorize the data

  ("reset, show Choc_2_Melter, group")

- checks for other applications which use data from the chocolate melter ("apps") and after seeing the familiar application named "AppMelterGraphs" invokes this canned application showing graphs of the melter's history ("reset, show AppMelterGraphs and Choc_2_Melter");

- saves this point of view named "Daily" ("save Daily"). Later in the week, after forgetting its name is reminded by asking for saved POV's from last week ("show Saved and LastWeek");

- checks unread mail ("reset, show Mail and unread and TimeLine"), and now de-emphasizes replies by himself ("less Me")

'Me' is an alias node, linking directly to a node representing his user ID;

- begins an email to co-worker ("reset, show 'John Smith', AppMail"). "AppMail" is a canned application which starts from the current point of view and searches the neighborhood for two data nodes: a data node directly connected to an abstract node that is equivalent to "Name', and another data node connected to the abstract node "Address". It then formats a text window for composing an email message; and

- enters appointment with Scharffen Berger chocolate supplier (input "mtg tomorrow 4pm 'Scharffen Berger'") and sees TimeLine with the event for April 28, 1999, more info about the contact person at Scharffen Berger is visible.

DataSea is a comprehensive program that stores, manipulates and visualizes all forms of data. Visually animated objects, or nodes, represent data or abstract concepts. Interactive commands (which something like verbs) operate on nodes and the links between them (which act something like nouns). These commands change internal parameters of the nodes and links. These parameters are visualized by qualities such as position and size. Certain nodes are emphasized, presenting information. The user finds the data or resource needed without knowledge of the data structure.

Unusual features of DataSea include relatively natural commands, robustness to imprecise queries, ability to generalize, absence of restrictive structure, use of semantic information and smooth transitions between visual states of the user interface. The simple commands and feedback from smooth visual transitions is key in integration DataSea with a voice interface.

The front-end of DataSea is a query interpreter and visualization system, and on the back-end is a database and API (application programming interface). Briefly, one sees a 'Sea of Data', and after each of a series of commands, one sees increasingly relevant data more prominently.

DataSea nodes act something like nouns of a natural language, and DataSea commands something like verbs. Here are the principal steps involved in a user query:

- Establish a point of view (either an existing node or a new, 'blank' one: if new, enter one or more reasonable values for broad, relevant terms of the query).

- Invoke commands (such as 'show', 'back', 'similar', 'abstractions') followed by more words of the query.

- Directly (e.g. 'more wordXXX') or indirectly (via commands such as 'group', 'similar', etc.) manipulate the presentation, progressively emphasizing information that is more relevant.

In a preferred embodiment, DataSea is a pure-Java application that can serve in a range of roles. It can view and control existing and legacy data such as email, documents, file directories and system utilities. It can ultimately serve as the principal UI to a system managing all data and system resources of a personal computer or workstation.

The natural ability of people to recognize visual patterns can be leveraged to convey information rapidly to the user. For instance, certain algorithms which depend on particular node and link configurations can render and position those nodes for rapid recognition. For example, if a target DN is surrounded by intermediate DNs which are themselves linked each to a distal AN, then those intermediate DNs are probably describing the target DN. The number of intermediates is then a measure of how much information is known about the target DN.

Its ability to gracefully reduce the complexity of the visual output means that a wireless hand-held client can be used to quickly browse and retrieve information from a remote server.

The simplicity of commands and accessibility of DataSea to the novice user lends itself to voice commands that can be used to navigate and control the display of DataSea.

The simplicity of DataSea's data structure allows easy acquisition and integration of legacy data into DataSea. Because new data is integrated with old, the acquisition of new data not only allows its retrieval by the user, but also enhances the user's retrieval of older data. Thus, as DataSea matures in its data content, queries are more robust to imprecise terms from the user. Since DataSea captures the information in the data and its structure from legacy databases, applications in DataSea can emulate legacy applications, while of course making this information available to broader use within DataSea.

While DataSea can emulate a RDBMS, without the complications of tables and foreign keys, the rich connections of DataSea and its ability to insert abstract nodes opens the way for neural-type processing. Learning-by-example is one example of that new capability. Learning-by-example refers to adjusting mag and CS values by `voting' (via `more' or `less', for example) on DNs, without relying on ANs. This selects DNs which the user especially likes or dislikes. Applying commands to DNs (such as files or URLs) changes not only the mag of each DN, but changes the CS and mag in its neighborhood, typically spreading through related ANs, thereby changing the mag of other DNs in the neighborhood, i.e., having similar qualities as the DNs that the user liked. A different point of view applied to DataSea, by virtue of different connections and connection strengths, changes the presentation of data as fundamentally as changing the database design in a relational database, but much more easily.

DataSea can be used to perform simple web-history viewing, data mining, and can be used as the principal Desktop UI for a computer showing all of the computer resources.

LEGACY DATA AND NETWORK TOOLS: Viewing domains such as file systems, web history or HTML documents and computer networks are obvious uses of DataSea and are early targets of DataSea. Applied to a web browser, the text of links to the current URL can be retrieved and parsed into DataSea, in effect pre-digesting it for the user.

VOICE INTERFACE TO WIRELESS HANDHELD DEVICES: The GUI (Graphical User Interface) of DataSea is important, but the underlying structure of DataSea queries and input methods are curiously appropriate for voice and natural language interfacing. Since queries, input and control of DataSea rely on simple words DataSea, current voice recognition software can be used instead of text input, and would significantly improve the uniqueness and general usability of DataSea. No other UI uses voice or is as appropriate for voice control. Since the results of many queries may be a short answer, voice generation is an appropriate output method, in addition to or instead of graphic output. For instance, the query 'show John Smith and address' is precise enough to generate one value significantly stronger than others, and therefore amenable to a programmatic decision for selecting which results to submit to voice output. In this way, voice can be a complete communications method, opening the door to remote access *via* telephone or wireless device.

PORTAL: Another opportunity involves selling server time for web searches, giving away client software initially. A typical interaction might involve throwing a number of search terms, asking for a display of abstraction categories or examples of URL's followed by the user judging prominent nodes, repeating as the search narrows.

DATA WAREHOUSING: DataSea's data structure and tools lend themselves naturally to data warehousing and mining, each with an estimated worldwide budget in 1999 of nearly $2 billion. DataSea intrinsically provides data mining and warehouse support. DataSea supports any type of data without specifying in advance the fields or tables to use. This is good for arbitrary user input, such as free-form notes, or machine-generated input, such as received data from automated test-equipment. DataSea therefore is a completely flexible data warehouse.

DATA MINING: Data mining is supported by DataSea's ability to reorganize any data based on user-defined point of views, the ability to link any and all data, and the ability to store the processing of data and applications into DataSea itself.

PRINCIPAL UI: DataSea can serve as the Desktop screen, the principal interface to all system services, independent of operating system. It can do this on demand, without locking the user into a particular operating system.

ARCHITECTURE OF DATASEA

Java Objects

The most used variables of object Node are Name, dist, mag and links[];.

Definitions:

```
global variables:  Node PointOfView_node, lastNode;

Class Node extends Object { // Important variables in
each node

 Object Data; // contains any computer representation
      of data, and includes get and set methods

 int dist; // the number of links from this node to
      the POV or another node.

 int tdist; // a temporary version of 'dist' used in
      calculating the minimum number of links from a
      node to other nodes.

 double mag, x,y,z; // x is the 'x' position in the
      DataSea coordinate system

 double px, py, pz; // px is pressure in x direction
      resulting from positioning routines

 double potentiation; // used to make node more
      sensitive to effects such as magnify

 TimeStamp potentiationTS; // time of last
```

```
     potentiation, used to degrade effect of
     potentiation as time passes

String Desc, Type; // used to describe the node. Type
     is typically DN, AN, Event

LinkObj links[];

}

Class DataObj extends Object { // contains any
computer representation of data, and includes get and
set Methods

String s;

...

getDataAsString().

...

}


Class LinkObj extends Object {

Node linked_node;

Double CS;

TimeStamp TS;
```

```
String Desc, Type; // used to describe the link, may
     refer to the source of the link, whether its an
     alias or not. usually set by the creator of the
     link.

}


Class VRObj extends Object { // VR stands for Virtual
Reality

double VRx, VRy, VRz; // relative positions in VR
     space, typically positions offsets from another
     node identified by recursive calling sequence or
     information contained in this or in related
     nodes or links.

boolean VRlocal, VRenabled;

VRShape Shape; // data and methods to render semi-
     realistically, for Virtual Reality presentation.

}


Class NameListObj extends Object {

// Acts as an index for all nodes.

// Vector, hashtable or other implementation of all
     nodes for rapid access based on name and or
```

```
    other fields such as TimeStamp and Desc

// METHODS

Node getNodeNamed(String s) {};

}
```

EXAMPLES OF SUBROUTINES USED IN PREFERRED EMBODIMENT

- Node.getChildCount() // return the count of distal links
- Node.getChild(int i) // returns i^th link with distance > Node's distance
- Node.getParent()
- Node.getNodeNamed(String s) // finds a node named 's' anywhere
- Node.getNearbyNodeNamed(String s, int max_distance, String type) // finds a node named 's' within 'max_distance' links of Node, having Type 'type'.
- Node.getConceptualDistanceTo (String s, int max_distance, String type) // returns result which is a function of distance to the target_node named 's', Type 'type' and the CS's to and including the target_node, and the mag of

target_node.

- Node.get/setNodeLinkedToAN(String an_name, Data data_value) get or set the value of the node between 'this' and AN(name)

- set_dist(Node starting_node) // recurses, calculates and sets Node.dist by finding the shortest route to each node by recursing from starting_node

- set_POV(Node target_node) // { set_dist(target_node); POV=target_node; }

- show (String name) { create_POV(); POV.link(getNodeNamed(string)); set_dist(POV); }

EXAMPLES OF USER COMMANDS

Most methods have three versions of arguments: (), (String s), and (Node n1, Node n2 ...). If null, then lastNode is used, if String, then matching nodes are looked for: both pass one or more nodes to the third version which takes explicit Nodes.

- **Show**(), (Node target) // link target to point of view, create point of view if necessary

- **Abs**() (Node target) // magnify distal ANs showing category of target (the AN is in a sense a category). An AN related to the target by two or more intermediate ANs will accumulate magnification via those intermediates. Follow distal paths, magnifying ANs along the way. Any AN along multiple distal paths will be magnified multiple times. Thus higher level ANs are emphasized.

- **Back**(Node target) // working proximally from target, increase mag of all until point of view is reached

- **And**(Node target_1, target_2) // potentiate neighborhood of target_1, then raise mag in neighborhood of target_2 if potentiated

- **More**(Node target) // raise mag in neighborhood of target, reducing the amount of change in mag as a function of spread_mode: e.g. proportional to the distance or a constant up to some threshold distance.

- **Potentiate**(Node target) // similar to More(), but the value of the variable potentiated is increased rather than the variable mag, and the potentiation TimeStamp ('potentiationTS') is updated and used to tell other routines when this was last potentiated. Typically other routines will reduce their modifications to variables as the elapsed time (currentTimeStamp-potentiationTS) increases.

- **Sim**(Node target) // indicates DNs which are similar to target based on their connections to ANs or other nodes. Similar to abs() but DNs on multiple paths are emphasized. Note: `abs' and `sim' use similar mechanisms traversing nodes. One emphasizes ANs resulting in abstracting the categories of the starting point, and the other emphasizes DNs thereby showing nodes that are similar to the starting point.

- **Group**(Node start, int target_level) { // group DNs around ANs which characterize them. From point of view, go distally until child.dist==target_level. If child is an AN, then force parent.X = child.X which clusters the data nodes between the start and the child abstract node onto the abstract node. Wait a second or so, letting the data nodes spread apart some, and repeat for (target_level--).

- **Recent**() // magnify nodes with recent TimeStamps.

The usage "Recent 1 hour" sets the value "1 hr" into the DN between DN(now) and AN(range).

where "range x = y" means:

```
(DN(now).getDNhavingANnamed(range)).setData("1
    hour")
```

We next describe some of the above-mentioned commands in another way, and we describe other commands:

DATA MANIPULATION COMMANDS

- **Show** links specific keyword to a point of view, zooms on it and emphasizes it and its neighbors.
- **Group** Starting from the point of view, secondary (2^0) data nodes spread distal magnify to directly connected abstract nodes, and set the secondary nodes position next to the largest directly connected abstract node, thereby grouping them.
- **Link | unlink** links specific nodes to a point of view or other nodes.
- **More | Less** emphasizes specific keywords given by the user and their immediate neighbors.
- **Abs(tractions)** emphasizes abstract nodes related to a data node, higher levels of abstractions being dominant initially.
- **Sim(ilarities)** emphasizes data nodes which are similar to a selected data node.
- **AND** emphasizes nodes near two or more selected nodes, similar to the boolean 'and' function,

although as with most aspects of DataSea, the result is not a binary decision. The non-linearity of the AND operation is adjustable, bringing in more or less of the neighbors. A highly non-linear mechanism akin to neural 'potentiation' can also be used, which can give very precise selectivity to the process of adjusting connection strengths and magnitudes.

- **SS** Spreadsheet simulation, given one data node, this presents related data nodes in tabular form with their principal abstract nodes as column-headers. Useful for tabular output.

- ***TL*** A fast synonym for "zoom TimeLine", "more now". Now' is a node updated automatically with the current time, linked to the TimeLine and nodes containing preferences for concepts such as 'recent'.

VIEWING COMMANDS

- **Back** emphasizes data nodes going backwards from a distant abstract node to the point of view.

- **Zoom** Centers and magnifies the screen image appropriately on a node or group of nodes

SUPPORTED APPLICATIONS

- ***Mail*** sends email to an address that is either explicitly selected, or begins a dialogue to choose one or more addresses based on their proximity to the current point of view. This is an example of a command which uses information from neighbor-values such as type and distance to make decisions. Uncertainty is resolved by the user who selects from a list of candidates proposed by the application.

- ***Simple Tabular Presentation (Spread-sheet format)***

- ***Activate*** runs the most appropriate program on a selected data node. Exactly which program is easily determined and changed if desired, since it is a functional node connected to the selected data node, and is thus viewable through normal DataSea techniques.

- **Input** takes text given by the user, parses, time-stamps and stores it into DataSea. A typical example of this would be *ad hoc* notes, such as 'phone-call from Bob about printer problem', or 'phone-number of Mary Smith is 845-1234'.

EXAMPLE APPLICATIONS

- **Mail** (Node target) From the target node, search the neighborhood for AN("name"), and use the DN proximal to it. From that DN, search for a DN connected to AN("address"). Similar for other ANs of use to a mail program.

- **Notes** Entire note is made into a DN, words become ANs with links to the parent DN.

The special syntax

"word1=word2"

creates AN(word1) linked to DN(word2).

- **SSheet**(Node target) A tabular representation of data and column headers of linked ANs in the neighborhood of target_node is built:

Collect all DNs linked to target. These represent one row of tabular presentation. Label these with column headers of the names of their directly linked, distal ANs. Each subsequent row is built from DNs linked to ANs and each other.

Set all of the VR position variables to appear in DataSea display as tabular format when in VRmode.

- **Phone**(Node target) looks in the neighborhood of target for a DN linked to AN("phone number")
- **Dir**(Node target) is a special case of SSheet, and looks specifically for directory-related information.

DATA ACQUISITION

Any application can store new data into DataSea, e.g. the applications Notes and Email.

Custom programs can translate legacy formats into DataSea linked nodes, e.g. to load information about a file system, the names of files and directories are stored into a tree representation first, then suffix and name can be used to create ANs, then content can be analyzed, e.g. by putting it through the Notes processor.

A RDB (Relational Database) would be loaded by storing the names of databases, tables and columns into ANs, and then values into DNs and keys into links. All these would be linked appropriately: e.g. table name linked to column names linked to all DNs having the values in those columns.

Web indices and browser histories can be stored.

System resources can be represented in DataSea.

A dictionary or synonym list can be loaded. The Type and Desc of links between synonyms or nodes with similar meaning are set. E.g. Type="synonym", Desc="from Webster's 10th Ed."

The user need not know about the data structure, such as database tables and their entity relationships in a relational database, or the directory structure of a file system. Nor does the user need to parameterize and decide how to store data, but may rather simply stuff it into DataSea. DataSea will parse the textual data and create links to representing abstract nodes. Abstract nodes are typically single words representing simple or complex concepts, and are linked to data nodes related to them. These nodes typically are massively linked.

DataSea is accessible from external programs via its API. More interestingly though, Java code may be stored into a node, fully integrating data and methods. The Java code can then act from within DataSea, for instance modifying the rendering of objects or analyzing data and creating new nodes and links.

The sequence of positioning and rendering flows through the network of nodes from the POV distally. Typically an application will start from one node, specified by name, pointing device or other means, and will search the neighborhood of that node for certain relationships or values and types. For example, invoking "Phone Jim" can find the nearest DN(Jim), then present the nearest DN which is linked to AN("phone number"). Thus commands like "Phone emergency" can work since 'emergency' can be linked to '911' which can have a large default CS which allows it to dominate, and "Phone 123 Main St" can work since the address "123 Main St" can be linked to a phone number through a DN of a person's name.

In addition to the DataSea commands such as show, abs and sim, new applications can be written to extend the base command set of DataSea.

All nodes have the capacity to store a VRObject which contains position and rendering information. It includes a triplet of numbers describing the relative position of a child to its parent, if the rendering mode of DataSea is set to 'VR-mode'.

APPLICATIONS IN DATASEA: HOW THEY DIFFER FROM TYPICAL APPLICATIONS OUTSIDE OF DATASEA

Typically computer applications use or set values *at specific locations* of memory and may or may not check their values by some means or rules or comparisons.

DataSea looks for information by nearness (a fuzzy metric) and/or characteristics of its links and/or characteristics of nodes directly or indirectly linked and/or their values.

Besides looking for DNs which are linked to specific ANs, an application in DataSea can query the distance or conceptual distance from a node to one or more values (of values such as data values, TimeStamps or other parameters). Decisions can be based on complex functions of environment checking.

VISUAL PRESENTATION

The visual tools of DataSea are based on a visual language which is completely different from today's standard GUI's and gives the user easier access to relevant data, and inhibits irrelevant data. DataSea can visually present large amounts of data and the relationships amongst them, emphasizing that which is relevant while keeping the larger context. The user sees exactly the data that is needed as well as related data, a form of look-ahead, albeit at lower resolution.

The data presentation changes as the user interacts with DataSea. Data moves smoothly from the background to the foreground, bringing it to the users' attention in response to the user. The gradual shift in visual states helps the user to understand what is happening as the query progresses.

The scene begins with a sea of objects representing nodes. Ordering of this sea begins as a result of commands to set a POV or by changing the mode to VRmode on some or all nodes. Typically one sees the sea of data in the background with the POV in the foreground and a TimeLine along an edge such as the bottom. Nodes move and change their appearance with interactions. These interactions can be with the user or with programs inside DataSea or externally.

The positions of nodes are changed by iterative calculations of forces on them, thus they move visibly between positions, rather than jumping suddenly. In this way changes in state, and thus appearance, can be followed by the user better than by sudden changes of appearance.

VISUALIZATION (IN ACCORDANCE WITH THE INVENTION)

Nodes are positioned dependent a set of pressures from sources, each pressure from a source (e.g. POV, parent, neighbors) being a function of that source's preferred position or distance between the child and the source, the child's mag, dist, etc. The optimum distance to point of view is proportional to dist/f(mag).

A node is stationary once these forces are balanced.

Rendering is also dependent on mag, dist, and mode.

Point of view is either a new temporary node set at a specific position on screen, or is an existing node.

Visual Presentation

The visual tools of DataSea are based on a visual language which is completely different from today's standard GUI's and gives the user easier access to relevant data, and inhibits irrelevant data. DataSea can visually present large amounts of data and the relationships amongst them, emphasizing that which is relevant while keeping the larger context. The user sees exactly the data that is needed as well as related data, a form of look-ahead, albeit at lower resolution.

The data presentation changes as the user interacts with DataSea. Data moves smoothly from the background to the foreground, bringing it to the users' attention in response to the user. The gradual shift in visual states helps the user to understand what is happening as the query progresses. For example, compare the ease of understanding either of these two scenarios: First, watching five animated objects, which represent five words in alphabetical order, reverse their order, representing reverse-alphabetical ordering: Second, watching five words on a line change from ascending alphabetical order to descending. In the first case, reversal is apparent. In the second, the simple operation of reversal is far less apparent: seeing the reversal requires re-analyzing the words and then trying out one or more possible explanations. In DataSea, nodes cluster and move individually and in groups in response to queries. Internal parameters inherent in each node and link change in response to queries. These internal parameters are mapped to visual behavior and appearances, such as size, position, color and shape.

These visual cues are used to enhance certain nodes or groups of nodes and their links. The internal parameters are changed by (typically recursive) commands that start at one node and spread through links to others. Commands adjust connection strength and magnitude of nodes based on their programmed algorithms and local node and link information, such as node type and the distance from the point of view. The point of view distance parameters are associated with each node and are functions of the shortest path from that node to the point of view. Recursive commands are self-terminating: typically but not always acting distal to the point of view (where the value of the next nodes distance is greater than or equal to the current distance) and often but not always producing less effect further away from the point of view.

The initial appearance of the GUI is a pseudo-3D view of:

- a backdrop containing the entire data set: The representations here are relatively stable, and provide an orienting reference for the user;

- a timeline along the bottom of the backdrop and

- a foreground region in which the user creates Points of Views (*point of views*) and into which data are brought forward from the backdrop. The dimension from back to front essentially represents the degree of customization of data presented to the user.

A new query is begun by entering words, similar to a web-search, or manipulating regions of the background with the mouse. One or more data nodes are directly `hit', increasing their magnitude, and secondary nodes (those distal to a primary) and their links are affected: exactly how depends on the spread mode of the operation. Nearness to the point of view is usually a function of link-distance and magnitude, but other methods are possible, e.g. link-distance alone which display data in a simple hierarchical set of 'levels'. Details of nodes are normally suppressed, but with the 'magnifier mode' turned on, any node under the cursor presents more information. Another mode is 'warp mode', which acts like a large magnifying lens on a region of the screen. This is similar to hyperbolic viewing of networks of nodes.

Which nodes are enhanced depends on the command and the spread mode, which is the way in which it traverses the linked nodes. The simplest spread mode is 'radial': this modifies the node at distance n+1 based on the strongest node directly connected to it of distance n, in effect being influenced by the node which is on the strongest path back to the point of view. Another spread-mode is 'sum', which adds up all the contributions of nodes of distance n to directly connected nodes of distance n+1. In 'sum' mode, a single data node distal to a large number of nodes will sum all their contributions. This is especially useful in the Similarity and Abstractions operations. If a specific node is specified in the query, it is enhanced by, for instance, growing in size and moving towards the point of view from the background blur of nodes. If an operation of an abstraction type is used, the abstract nodes are enhanced. The relative positioning of higher or lower levels of abstraction depends on the specific command. If an operation of a similarity type is used, data nodes predominate by approaching the point of view and by being enhanced.

Rather than connecting the hits immediately and directly to the point of view, abstract-nodes in common are first drawn near the point of view. These more abstract nodes are then followed by more detailed ones receding back to the backdrop, positioned to give the sense of their being pulled out of the DataSea. Qualities like time since an event, or distance to one or more chosen abstract nodes can act as a secondary force, or wind, acting to influence the position of nodes along one of the 3 dimensions of the visualization.

LINKS

Data in DataSea is heavily linked without restrictions on what can be linked. DataSea solves the 'cobweb' visual problem by establishing a **point of view** for the users' queries. The problem of following links that are loops is solved by calculating, on the fly, the shortest number of links from the point of view to the nodes. This turns a series of self-referencing loops into a temporary hierarchy, based on the current point of view.

The user can browse raw data in DataSea, but meaningful structure comes from the interaction between the point of view and raw data. This is analogous to the quantum-physics effect of forcing a wave function into a specific physical state by applying an observation to the wave function: interaction with the user that forces data into its useful, visible state.

A point of view is one form of an abstract node. Once the user finishes a query, the point of view that has been created can be absorbed into DataSea and used later, a form of 'checkpoint' used in calculations.

Links can occur rather mindlessly, for instance simply by association to part or all of an inputted document, in a way which captures relationships, for instance field definitions from legacy databases, or semantic meaning from, for example, some level of natural language processing.

Postprocessing inside DataSea creates abstract nodes. These represent abstractions of the data inside DataSea, representing concepts or the results of analysis. A 'mature' DataSea will contain a large proportion of these abstract nodes.

Each event which links data within DataSea stores a link ID along with it. Thus any two nodes can be linked together more than once, each link having a different ID to differentiate the context of their being linked. A single link ID can be used between many nodes, as long as that particular subset of nodes has a meaningful context. This context is stored in an abstract node which, linked of course to the subset with that link ID, and contains the reason for the links.

DATA

Data is user-defined and customizable: whatever the user puts into DataSea, it merely needs to be in a computer representation. Data is held inside so-called 'nodes', which may be linked together. A data-node can be a specific value, text such as a web page or free-form entry, or an object representing something as complex as a virtual-reality view of a manufacturing facility. Text in any language is broken up into words and stored. All of the different forms of data share identical mechanisms of storage, linkage, search, presentation and access. The database contains highly linked data but differs in significant ways from RDBMS's (relational database management systems), including the ability to create links between any data and the elimination of structured tables. Rather than using pre-defined fields to capture relationships, DataSea uses nodes with appropriate links. As new data is introduced and linked to the existing nodes, alternate paths are created between points. This allows data to be found which contains no keywords contained in the query, relying on associations contained in the new data. A simple example would be loading a dictionary into DataSea: there are few related concepts that are not linked through only even two or three definitions of either. Thus, a user may enter a query containing no keywords of a document and be presented with that document, albeit emphasized less than documents that contain more direct links to the query terms. AI or manual 'digestion' of information and linkage to abstract concepts is of course possible, as is done by those who compile databases for search engines today.

The user need not know about the data structure, such as database tables and their entity relationships in a relational database, or the directory structure of a file system. Nor does the user need to parameterize and decide how to store data, but may rather simply stuff it into DataSea. DataSea will parse the textual data and create links to representing abstract nodes. Abstract nodes are typically single words representing simple or complex concepts, and are linked to data nodes related to them. These nodes typically are massively linked.

DataSea is accessible from external programs via its API. More interestingly though, Java code may be stored into a node, fully integrating data and methods. The Java code can then act from within DataSea, for instance modifying the rendering of objects or analyzing data and creating new nodes and links.

APPLICATIONS

A fully integrated application in DataSea uses the DataSea linkage and VR mechanisms to provide the functionality of typical window/menu systems. The program of the application is stored in a DataSea application node.

- The typical steps taken by DataSea applications include:

- the neighborhood of the target node is searched by the application for application-specific data requirements

- New formatting nodes are created (eg. A 'page' representing the template for a letter)

- Links are made to data nodes and their VR positions are set relative to the formatting nodes.

VR MECHANISMS

All nodes have the capacity to store a 3-D vector called a 'VR-position'. This is a triplet of numbers describing the relative position of a child to its parent, if the rendering mode of DataSea is set to 'VR-mode'. Any child having non-zero a VR-position variable will position itself relative to the calling parent based on the VR-position values.

ESSENTIAL INTERNAL ELEMENTS OF DATASEA

In a preferred implementation, DataSea is a pure-Java application. Once loaded, user-defined data-nodes and links are used to visualize information from a range of sources in an interactive or programmatic way. Data-node sources can be email, web sites, databases or whatever is required. Fig. 3 is a block diagram of an embodiment of the invention. All data is contained in objects called nodes. Information describing the data is held in the data-node. A complex data-node may be broken into smaller ones. A data-node has a set of standard fields describing itself and any number of links to other data-nodes.

The DataSea database is a highly linked structure of nodes. A link contains information describing itself and how it relates the linked data-nodes. It therefore contains *semantic* information, adding a new dimension to interactive or programmed processing of data. That is, DataSea supports not just parametric searches (which find the values at certain storage locations specified by parameters) or content-based analysis (which find particular values and their relations anywhere in the database), but the *meaning* of a collection of nodes. An example of this could be a link with the description "located near" relating a computer with a person's name.

Processing of data occurs not only on values of certain parameters, but on any value, independent of what it is describing. For instance, one may search for all information related to an individual's name without specifying which table and column of the database to search, and in which tables and columns to look for foreign keys.

Applications can run inside DataSea, in fact these applications are themselves held inside a node. Current applications such as automatic report generators and data formatters, know which pre-defined data fields to place, just where, and how to order the values. This functionality is served by DataSea's mechanism of node and link descriptors, which can act as the column names of RDBMS's. The DataSea link description however also provides semantic information about those relationships.

Objects are positioned and rendered strongly dependent on their content and their links. That is, features of the rendering of nodes and the relative positions of nodes depend on content and links. Thus, DataSea is unique because the presentation is strongly dependent on the data itself.

USAGE SCENARIO

Below is a scenario of events with comparisons between two different application approaches: The user routinely stores information and calls it up later when faced with a decision as to repair a new printer or buy an old one. This example shows the simplicity and time saved with DataSea. It compares:

1) DataSea, and

2) A mix of applications consisting of "Outlook Express", "Excel", and "Internet Explorer" in a Microsoft Office Suit, along with two other applications, "Tracker", a call-tracking application and a database front-end application called DB-Front-End here. The numbers of seconds in parentheses following these two methods are estimates of time needed by the methods, in addition to the event itself.

Event: The user receives email from a friend who mentions her new H.P. printer	
Office Suite: email is stored in Outlook Express.	**DataSea**: email headers and text are automatically stuffed into DataSea.
Event: The user surfs the web and finds advertisement for HP Printer	
Office Suite: Internet Explorer saves the non-overlapping history of URLs temporarily, and relies on the user to bookmark special URLs, and put them in the tree hierarchy defined by the user.	**DataSea**: with links to DataSea from the browser, each URL visited is stored into DataSea.
Event: The user gets a phone call and makes a note to himself that repairman Bob Smith says that printer A will cost $300 to repair, and that it is in Joe Baker's office.	
Office Suite: The user opens the call tracking program 'Tracker' and fills in the	**DataSea**: The text of the note is stuffed into DataSea, and explicitly enters: "(Printer A)

fields prompted by the wizard, including the note text "Repairman Bob Smith called...". To store the location of Printer A in a company-wide database, the user invokes the database editing application DB-Front-End, selects appropriate view (e.g. Machine_View), searches for 'Printer A', enters 'Joe Baker' for the column 'Location'.	(office=Joe Baker)". The information is parsed and time-stamped automatically.
Event: The user now wonders if he should replace printer A with a new one. He remembers seeing a reference in a recent email for an HP printer, and also an HP ad on the web, but can't remember exactly where he filed this information.	
Office Suite: User opens Outlook Express, tries to recall the name of the email sender, possibly keywords to search and sets the time range to search (enlarged since an event 1 minute outside the range will be excluded). Immediately he sees messages focussed on one keyword. User skims header and text to decide if this is the correct message. Then, user opens Internet Explorer and browses the names in the History list, trying to recall the context for each as he sees them, or tries to recall the name of the document corresponding to the right URL. He then deduces which database stored procedure, table or view to use, opens the DB-Front-End application, enters 'Joe Baker' in the correct search field and sees "Printer A" in the Equipment column. He arranges the four windows from these applications for simultaneous viewing (Outlook, Explorer, Tracker, and DB-Front-End).	**DataSea**: User starts a point-of-view with the initial associative words '"Joe Baker", Printer, email' and gives his guess of when this all occurred via mouse drag on the time-line. He sees several concept-nodes and some data-nodes. He then judges these by emphasizing/de-emphasizing particular ones, and sees email with appropriate links. He further judges them, adds the word "URL" to the point of view which results in the appropriate URL and data being pulled forward

<u>Example (Sales Pitch): Laptop starts DataSea, voice interface enabled</u>.

User says '*show Mail'* ... mail nodes swell, abstract nodes visible.

User says 'show *Files'* 'Tax 97' brings directories forward and shows files.

User says '*show John Smith'* which crates a point of view, linked to abstract node named "John Smith"

User says '*BMW'* which shows Smith's "BMW 528 1985" by virtue of abstract node "car" linked to "BMW", "Ford".

User says '*show address'* bringing "123 Main St." forward.

User says 'reset', then 'show address' and sees names and addresses of all entries.

User marks timeline over the past week, and says 'show printer and email and Hewlett Packard' which shows an abstract node "Printer" linked to email message about printers and a web page of HP printers

User says 'input "*John Smith telephone 848-1234"* which creates a node holding the entire message, and parses it into smaller data nodes.

User says '*show John Smith'*; one sees his telephone number.

To demonstrate abstract nodes and learning: have processed 50 URLs from 'cat' web search,. See all 50 around the abstract nodes surrounding the point of view named 'show cat'. User deselects URLs not related to technical descriptions, the abstract nodes change, bringing forward URLs with more technical information.

BENEFITS

1) Immediate visualization of user-defined data.

2) Quick visual feedback on relevant data.

3) Less time required to interpret complex data.

4) Higher user productivity because DataSea is an intelligent organizer of data.

5) Non-technical user can view data in way they understand, not the way the database may be organized.

6) Reduces dependency on programmers.

7) Reduced bug-count and time for programmers.

8) Simpler usage model through single tool to manage and visualize information.

9) Time is saved in storing and retrieving information.

10) Databases can be joined automatically without custom code.

11) Points of view can themselves be stored into DataSea, storing interactions with the computer.

12) Queries and processing results can be stored into DataSea, and used as any other data.

13) DataSea learns by example: The user may search for data based on relationships to known data or high-level concepts. Judging can be applied to specific data, such as documents, or to concepts, resulting in `learning by example': the mechanisms of positive and negative feedback to the system are the same.

14) New data is automatically integrated: New data can be entered and automatically integrated, allowing non-programmers to store data without adapting to the database.

15) Interoperability issues are moot: Programs can be integrated into DataSea as well as simple data. Since all links between nodes use the same mechanisms, any program has access to any data.

16) All data can be viewed while maintaining orientation and context: The user can always quickly orient themselves, sparing confusion because data is viewed from the point of view that the user has designated. Context is maintained by position and rendering cues, which

indicate the sources of the data and their immediate relationships. The background with its clusters of data-nodes is relatively stable and familiar, and as data is pulled out from it towards the foreground point of view, the data's position is influenced more and more strongly by the criteria of the point of view and nodes connected strongly to it. The user `judges' nodes: emphasizing a node will enlarge it and bring it to a more noticeable position.

Fig. 4 is a screen-shot which shows the result of entering the simple command `show rocky' ("rocky" representing the name of a user, who has previously entered data pertaining to himself into the system).

Fig. 5 is a screen-shot which shows the result of entering `abs' (when the Fig. 4 display has been generated), bringing forward the abstract nodes which are distal to the data node `Rocky'. Note the abstract node `Directory', which, because it groups the abstract nodes `phone' and `address' and is thus at a higher level of abstraction, is positioned closer to the point of view than `phone' and `address' abstract nodes.

Fig. 6 is a screen-shot which shows the result of entering `back phone' (when the Fig. 4 display has been generated), bringing the data node between the abstract node `phone' and the target node `Rocky' forward.

Fig. 7 is a screen-shot which shows the result of entering `SS' (when the Fig. 4 display has been generated), which gives a simple 2-column spreadsheet based on the current target data node `Rocky'.

Fig. 8 is a screen-shot which shows the result of entering `show egg-tempera'. It shows the primary abstract nodes. But what if we want to see some examples of data nodes which are similar to `Egg-Tempera'? If so, one could enter the command `show egg-tempera', `similar', resulting in the screen-shot of Fig. 9. It is apparent by comparing Fig. 9 with Fig. 8 that the node "Frescoes" and nodes "Glazing_techniques", "acrylics" and "oils" are brought forward in Fig. 9, near the target data node "Egg-Tempera".

Aspects of the invention include the following:

1. Methods of automatically creating a highly connected network of nodes containing data from computer-readable sources. Information contained in the structure of legacy databases is captured. All data can be integrated. The nodes are identical in structure, as are their links, differing only in their content.

2. Methods to interactively explore, access and visualize information in a highly connected

network of nodes. These involve setting a point of view, linking some number of nodes directly to it and calculating individual link distances from all data nodes back to the point of view. This creates a hierarchical network amenable to visualization even though there may be cyclic loops in the links. This hierarchy may change whenever a link is added or deleted. Other internal parameters such as the connection strength of each link and the magnitude of each node are used in the visualization to calculate position and size of each node.

These methods:

A. emphasize relevant data throughout the query process;

B. are tolerant to imprecision and errors in queries. This ability improves as the data set grows;

C. allow access directly, or indirectly; retrieving relevant data containing none of the key-words used in the query;

D. allow finding data similar to known data, without specifying its characteristics;

E. give smooth changes in visual state rather than step-wise changes, and provide information to the user in the manner that the nodes move (speed and direction) and appear (size, color);

F. show available categories that a particular datum is a member of;

G. integrate virtual reality renderings when appropriate;

3. Method of breaking display space into an array of cells, having dimension one more than the dimension of the space displayed on the screen, the extra dimension being size. These are linked to nodes and used by the user interface to rapidly access individual or groups of nodes.

<u>Additionally these methods</u>:

A. are accessible to the naïve user;

B. allow emulation of applications such as relational databases and spreadsheets;

C. use a simple command and query syntax which is amenable to a voice interface;

D. use time efficiently: user spends time using commands that act directly on data, rather than time spent navigating a pull-down menu interface.

E. focus time spent on becoming expert on the data set, rather than the user interface.

Variations on the preferred embodiment include:

Variation 1: Voice integration. Front end routines take either keyboard input or voice input, submitting word strings from either to handler functions. Voice word 'go' acts as keyboard 'Enter'.

Variation 2: Client server, a wireless or wired client, display mode set to abbreviate early.

Self Diagnostics and Use as a Debugger:

DataSea can be used to visualize the DataSea program itself. Besides visualizing nodes which represent data for the user, as described elsewhere in this document, in so-called `dataset nodes' the nodes that are visualized in DataSea can represent internal programming objects, methods or elements of DataSea itself (providing a sort of built-in debugger).

Code can be inserted into the program which will visualize each method's invocation and its modifications of user data.

DataSea separates the two tasks of modifying the values of node variables and rendering of those nodes. Thus DataSea can redraw the entire scene not only after traversing the linked nodes and re-calculating their internal parameters, but the entire scene can be re-drawn at any time during these calculations, even once every time a dataset-node variable such as `mag' is changed.

Thus, a self-node can indicate to the user its own activity, by redrawing the entire scene normally and then highlighting itself, or drawing lines to a dataset-node or its elements that it is operating on.

For instance, if a user commands DataSea to increase the variable `mag' of a node, the method which does that (e.g., 'spread()') can draw a line from the self-node representing 'spread()' to the dataset node it is modifying. A simple implementation could be as follows:

If the method spread() recursively calls the method spread_recursive(), insert a conditional call to touch() after spread_recursive:

```
spread(Node node) {

  // for all children of node

  // spread_recursive(child);
// touch(node, child)   }
```

where `touch(Node caller, Node target)' will visualize the accessing and setting of variables in the target, where 'caller' is the spread() self-node and `target' is the dataset node being operated on.

The method touch(Node caller, Node target) could be implemented as follows:

```
touch(Node caller, Node target) { // Show a line
between caller and target nodes

 clear_screen(); // clear the screen

 render_all(); // render all the nodes normally

 draw_line_between_nodes( caller, target ); // draw a
                                                line

 sleep(500); // pause so user can follow what is

                    happening }.
```

Aspects of the invention include:

a method and apparatus for creating nodes containing data, linking the nodes into a network, setting parameters of the nodes (node variables, and maintaining information specific to each node, e.g. mag, CS, direction of the link (polarization). Each node preferably has a name associated with which it can be searched from a master list;

a method and apparatus using "context nodes" to modulate link connection strength (CS) and establish context for groups of nodes. For example, a method for associating a set of links and establishing a context node which can modulate the CS of those links thereby sensitizing or desensitizing them to further operations. The context node can also magnify the nodes linked by each link it modulates;

a method and apparatus for loading data from free-form notes. For example, a method of taking text input (text from user or application, or text resulting from voice translation) and establishing a set of linked nodes therefrom by:

creating a new node for the full text called the full-text-node;

discarding selected words (e.g. articles)

linking the full-text-node to individual nodes representing each remaining word in the full text, creating new nodes as needed.

For another example, a method of converting tabular data, i.e., text organized into rows and columns, with column headings (or RDMBS data, with additional links for the keys of the RDBMS), into a set of linked nodes, in which:

each column heading is represented by an AN, the column-heading-AN

each cell of data is represented by a DN

links are established between each column-heading-AN (representing a particular column) and those nodes corresponding to the cells in that column

links are established between those nodes corresponding to each cell in a row from the table.

For another example, a method of converting

files from a computer file system or a set of files linked by HTML references into a set of linked nodes, in which:

DNs are established representing each directory or file;

links from each node are established to terms found in the file content, e.g., as is done in the parsing of notes. The procedure can filter the content looking for only certain tag values such as meta-tags or heading values (e.g. <H1> Title Here </H1> has "Title Here" as heading-level-1 in HTML)

Another set of links can be made to ANs representing the suffix of files, or such ANs can be used as ContextNodes for all links to those files.

For HTML files, links are to be established between nodes representing HTML files and other nodes representing HTML files that are referenced by the first HTML file.

For another example, a method of converting files from a computer file system, in which links are established between DNs representing file directory with DNs representing files or sub-directories in that directory;

a method and apparatus for defining a POV (either a particular node or a new node linked to a particular node);

a method and apparatus for defining distance (as a function of the number of links between nodes and the node type) and hierarchy from the POV and determining distal and proximal directions, in which: once a POV is set and distances calculated from it, a hierarchical 'tree' is defined from what was an arbitrarily complex cross-linked network of nodes. Thus, if any node 'x' has had its distance set by this routine, one is guaranteed to find a path from that node 'x' back to the POV by traveling on a path between nodes of ever-decreasing distance values;

a method and apparatus for retrieving data which is linked into a network of nodes interacting with the user to better present the desired data;.

a method for emphasizing nodes and paths by tracing backwards from a target node to a POV by following all links to nodes whereby the next node has magnitude less than that of the prior node. Emphasizing those nodes on the path(s) shows nodes 'between' the target and POV. By traveling backwards from the target node to the POV, there may be more than one node having a distance less than the target. This is fine, and if all paths backwards (with the requirement that they are consistently proximal) are emphasized it is fine. For example, with Bob being the POV, and traveling backwards from the node representing January 1999, all nodes such as notes and events related to Bob will be emphasized;

a method for assigning position to each node which is dependent on the node's parameter values, including distance, CS and magnitude. Rather than setting the node at the calculated position immediately, it moves there gradually thereby showing the transition between states. One way to do this is to calculate forces on a node which are related to the difference between the node's current position' and an ideal calculated position.

The ideal position depends on the positioning mode in effect:

a Relations Mode: Most suitable for narrow queries where we wish to see all the links between nodes in the target data set. Nodes fan out from their parent; the angle dependent on the number of children their parent has, their distance dependent either on (mag) or (1/mag); or

a Levels Mode: Most suitable for broad queries where there are too many links between nodes in the target data set. Starting in the center of the screen, fanning out to the left dependent on their distance from the POV, ANs are rendered. Starting in the center and belonging to the right half of the screen are the DNs whose position moves further to the right the lower their mag;

a method and apparatus for visualizing data, by appearance on a screen. For example, a method of assigning visual emphasis (color, size) to each node dependent on the nodes distance, CS and magnitude.

Examples of operations performed on nodes of the inventive set of linked nodes (or on a sea of displayed representations of such nodes) include:

'ABS'(for characterizing and understanding the environment of nodes and their ANs): from a target node, traveling distally and upstream, find the first AN and emphasize it. This 'abstracts' the target node in terms of linked ANs. To abstract it at a higher level, go from those ANs to directly linked ANs which are both distal and upstream. This can continue to arbitrary level until we run out of nodes (realistically not very far, a handful of levels);

'XABS' (for emphasizing ANs from a group of nodes, those ANs not having been recently visited by query operations: emphasizing distally from these ANs will result in a relatively large number of DNs being modified. The user may find ANs which are obviously related or not related to their interest, and thereby significantly change the presented data set. Since these ANs haven't been used recently, we in effect triangulate the target data set from more vantage points. Determining categories which, when evaluated by the user as good or bad, have a large effect on narrowing the presented data set, that is helping the user find the target data set;

'SIM': a method of emphasizing (magnifying) nodes based on their similarity to a chosen node without specifying values of any node (using the 'sim' command which emphasizes DNs linked to any or all of the ANs which are linked to the chosen node(s));

"POTMAG": a method of modifying the variable Potentiation of a node, and using that value to influence the degree of change to the variable 'mag' from a subsequent operation. Thus, one operation on the first set of nodes may call Node1.setPotentiationValue() and a subsequent operation on the second set of nodes may set the value of Node1.mag based on Node1.getPotentiationValue(). This 'primes' a set of nodes, and can operate approximately as a soft, or non-binary AND operation.

Another aspect of the invention is structuring of a set of linked nodes (a "network") including "application nodes" (sometimes referred to as applications). Applications are nodes containing code which get the information they operate on from traversing the network. E.g. an email node-application is linked to, or given a reference to, the node "Bob Smith", and upon being invoked (by the action function inherent in each node or otherwise) searches the neighborhood of the "Bob Smith" node for a DN linked to an AN representing email address. If more than one is found, the user is presented with the selection to choose from. Thus any node-application can be 'applied' to any node.

One aspect of the invention is a method of accessing data, wherein the data is structured as a set of linked nodes, and each of the nodes includes at least one link to another one of the nodes. The method includes the steps of:

preliminary to displaying representations of the nodes on a screen in a screen space having N dimensions, where N is an integer, dividing a display space having dimension N + 1 into an array of cells, wherein the dimension of the display space includes a size dimension;

linking each of the nodes to at least one of the cells; and

implementing a user interface which displays representations of at least some of the nodes on the screen having sizes determined by the cells to which said at least some of the nodes are linked, wherein the user interface rapidly accesses individual ones or groups of the nodes in response to selection of at least one of said representations.

What follows is a source code listing (in the Java programming language) of a computer program for programming a computer to implement an embodiment of the invention. In the listing (which consists of parts labeled "TL.java," "Timer.java," "ColorObj.java," "Link.java," "Mode.java," "Node.java," "Force.java," "GetURLInfo.java," "Input.java," "Populate.java," "GUI.java," "DataSea.java," "LinkObj.java," "VRObj.java," and "nsr.java"), the object "gui" of class "GUI" is the top-level object, and instantiates the object "datasea" of class "DataSea" (and other objects).

```
// This is TL.java     by Rocky Nevin

import java.lang.*;
import java.awt.*;
import java.util.*;
import java.sql.*;  // for the class Timestamp
import java.io.*;

/*
 * class TL (TimeLine)
 * init() initializes String month_names[]
 * create_date_node() takes date string, finds or creates&links smallest
appropriate time-node
 */

class TL extends Object {

static String month_names[];
static String day_names[];
static String hour_names[];
static String year_names[];
static GUI gui;

public void init (GUI passed_gui) {

gui = passed_gui;

month_names = new String[12];
day_names = new String[31];
hour_names = new String[24];
year_names = new String[10];

month_names[0]="Jan"; month_names[1]="Feb"; month_names[2]="Mar";
month_names[3]="Apr"; month_names[4]="May"; month_names[5]="Jun";
month_names[6]="Jul"; month_names[7]="Aug"; month_names[8]="Sep";
month_names[9]="Oct"; month_names[10]="Nov"; month_names[11]="Dec";

System.out.println("TL.init() done.");
return;
}

/*
 * month_from_double
 *            return string of month based on double between 0 and 1
 */
public String month_from_double (double val) {
if ((0>val) || (1.0 < val)) {
        System.out.println("month_from_double() Error:
month_from_double("+val+")");
        return("[month_from_double() Error: month_from_double("+val+")]");
        }

val *= 12;
val *= 0.9999; // 0 OK, 12 not OK
int m = (int)val;
return(month_names[m]);
}


/*
 * create_TS
 *              return Node based on String s linked to caller
 */
public Node create_TS (Node caller, String CNodeName, String s) {
        Node ts_node = create_date_node(s);
        Node CNode = gui.datasea.find_node_named(CNodeName, "CN");
        if (CNode == null) {
                //System.err.println("create_TS() Error: null CNode for caller
<"+caller.Name+">, string = <"+s+">");
                caller.link(ts_node);
                if (caller == ts_node)
                        System.out.println("create_date_node ERROR(1): caller ==
ts_node <"+ts_node.Name+">");
                }
        else {
                caller.link(ts_node, CNode, "polarized");
                if (caller == ts_node)
                        System.out.println("create_date_node ERROR(2): caller ==
ts_node <"+ts_node.Name+">");
                }
return(ts_node);
} // end create_TS


/**
 **  find_day_in_string  return a string which is a date, like 1,2...31
 **
*/
public String find_day_in_string (String s) {
int i, size;
String numbers =        " 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 ";
        numbers = numbers+" 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31";

String[] words = gui.input.string_to_array(s);
size = words.length;
for (i=0; i<size; i++) {
        if (0<=numbers.indexOf(words[i]))
                return(words[i]);
        }
```

```
return((String)null);
} // end find_day_in_string


/*
 * create_date_node
 *              search for year, month, day, time, create node for particular
date
 *              using finest resolution appropriate, create nodes for time
foundation
 *              -create day|month|year offsets
 *              -create lowest|highest resolution nodes, link them
 */
public Node create_date_node(String s) {
double x=0, year_offset=0, month_offset=0, day_offset=0, day_value=0;
boolean year_found=false, month_found=false, day_found=false;
Node year_node=null, month_node=null, day_node=null;
Node finest_grain_node=null; // The finest-grained TL node to which we'll link
DN to
Node coarsest_grain_node=null; // The largest-grained TL node to which we'll
link TimeLine to
Node last_node=null;
String year_str="";
String month_str="";
String day_str="";
String ret_str="";
double TimeLineSizeY = gui.datasea.pop.TimeLine.size_Y;
double TimeLineSizeX = gui.datasea.pop.TimeLine.size_X;
double TimeLineX = gui.datasea.pop.TimeLine.X;
double TimeLineY = gui.datasea.pop.TimeLine.Y;
double base_year_offset = 1.0/2;
double base_month_offset = base_year_offset/12;
double base_day_offset = base_month_offset/31;
double default_size_X = 2;
int delta_x = 2; // used as offset between year, month, day

//------------------------------------------------------------
// LOOK FOR XEAR
if (0 <= s.indexOf("1999")) {
        year_str = "1999";
        year_offset = 0;
        year_offset = base_year_offset * 0 * TimeLineSizeY;
        year_found = true;
        }
if (0 <= s.indexOf("2000")) {
        year_str = "2000";
        year_offset = base_year_offset * 1 * TimeLineSizeY;
        year_found = true;
        }
//------------------------------------------------------------
```

```
// LOOK FOR MONTH
if (0 <= s.indexOf("Jan")) {
        month_offset = base_month_offset*0 * TimeLineSizeY;
        month_str = "Jan";
        month_found = true;
        }
if (0 <= s.indexOf("Feb")) {
        month_offset = base_month_offset*1 * TimeLineSizeY;
        month_str = "Feb";
        month_found = true;
        }
if (0 <= s.indexOf("Mar")) {
        month_offset = base_month_offset*2 * TimeLineSizeY;
        month_str = "Mar";
        month_found = true;
        }
if (0 <= s.indexOf("Apr")) {
        month_offset = base_month_offset*3 * TimeLineSizeY;
        month_str = "Apr";
        month_found = true;
        }
if (0 <= s.indexOf("May")) {
        month_offset = base_month_offset*4 * TimeLineSizeY;
        month_str = "May";
        month_found = true;
        }
if (0 <= s.indexOf("Jun")) {
        month_offset = base_month_offset*5 * TimeLineSizeY;
        month_str = "Jun";
        month_found = true;
        }
if (0 <= s.indexOf("Jul")) {
        month_offset = base_month_offset*6 * TimeLineSizeY;
        month_str = "Jul";
        month_found = true;
        }
if (0 <= s.indexOf("Aug")) {
        month_offset = base_month_offset*7 * TimeLineSizeY;
        month_str = "Aug";
        month_found = true;
        }
if (0 <= s.indexOf("Sep")) {
        month_offset = base_month_offset*8 * TimeLineSizeY;
        month_str = "Sep";
        month_found = true;
        }
if (0 <= s.indexOf("Oct")) {
        month_offset = base_month_offset*9 * TimeLineSizeY;
        month_str = "Oct";
        month_found = true;
        }
if (0 <= s.indexOf("Nov")) {
        month_offset = base_month_offset*10 * TimeLineSizeY;
        month_str = "Nov";
        month_found = true;
        }
if (0 <= s.indexOf("Dec")) {
        month_offset = base_month_offset*11 * TimeLineSizeY;
        month_str = "Dec";
        month_found = true;
        }
//----------------------------------------------------------
// LOOK FOR DAY
if ((day_str = find_day_in_string(s)) != null) {
        day_value =
(double)(GUI.java_lang_double.valueOf(day_str)).doubleValue();
        day_offset = base_day_offset * (day_value-1) * TimeLineSizeY;
        day_found = true;
        }
//----------------------------------------------------------
year_offset += TimeLineY;
month_offset += year_offset;
day_offset += month_offset;
//----------------------------------------------------------

if (day_found) {
        ret_str = ret_str + day_str+ " ";
        if (day_value == 2 && month_str=="Mar")
                GUI.P(0,"create_date_node","---> day 2 for
<"+day_str+":"+month_str+":"+year_str+">");
        day_node = gui.datasea.pop.create_node(day_str+" "+month_str+"
"+year_str,"Event");
        //day_node = gui.datasea.pop.create_node_forced(day_str+" "+month_str+"
"+year_str,"Event");
        day_node.Y = day_offset;
        last_node = day_node;
        finest_grain_node = day_node;
        coarsest_grain_node = day_node;
        }
if (month_found) {
        ret_str = ret_str + month_str+ " ";
        month_node = gui.datasea.pop.create_node(month_str+" "+year_str,"Event");
        if (last_node != null) {
                month_node.link(last_node);
                if (last_node == month_node)
                        System.out.println("create_date_node ERROR: last_node ==
month_node <"+month_node.Name+">");
                }
        month_node.Y = month_offset;
        last_node = month_node;
```

```
            if (finest_grain_node == null)
                    finest_grain_node = month_node;
            coarsest_grain_node = month_node;
            }
    if (year_found) {
            ret_str = ret_str + year_str;
            year_node = gui.datasea.pop.create_node(year_str,"Event");
            if (last_node != null) {
                    year_node.link(last_node);
                    if (last_node == year_node)
                            System.out.println("create_date_node ERROR: last_node ==
year_node <"+year_node.Name+">");
                    }
            year_node.Y = year_offset;
            last_node = year_node;
            if (finest_grain_node == null)
                    finest_grain_node = year_node;
            coarsest_grain_node = year_node;
            }

    if (year_node != null) {
            year_node.X = TimeLineX+delta_x;
            year_node.size_X = default_size_X;
            year_node.size_Y = 0.95 * TimeLineSizeY * base_year_offset;
            }
    if (month_node != null) {
            month_node.X = TimeLineX+2*delta_x;
            month_node.size_X = default_size_X;
            month_node.size_Y = 0.95 * TimeLineSizeY * base_month_offset;
            }
    if (day_node != null) {
            day_node.X = TimeLineX+3*delta_x; // + (31-day_value);
            day_node.size_X = default_size_X;
            day_node.size_Y = 0.95 * TimeLineSizeY * base_day_offset;
            // day_node.Y += (Math.random()-0.5);
            }

    Node event_node = gui.datasea.pop.create_node(ret_str, "Event");
    event_node.X = finest_grain_node.X+2;
    event_node.Y = finest_grain_node.Y;
    event_node.size_X = default_size_X;
    event_node.size_Y = 0.95 * TimeLineSizeY * base_day_offset;

    // link to finest grain time node ...

    if (event_node == finest_grain_node)
            ; //System.out.println("create_date_node ERROR: event_node ==
    finest_grain_node <"+finest_grain_node.Name+">");
    else
            event_node.link(finest_grain_node);
```

```
// link to TimeLine node ...
gui.datasea.pop.TimeLine.link(coarsest_grain_node);

return(event_node);
} // end create_date_node

} // end class TL (TimeLine)
```

```
// This is Timer.java     by Rocky Nevin


/*
 * class Timer
 */
class Timer extends Object {
long TS1=0, TS2=0;
String TS1_name, TS2_name;


// CONSTRUCTOR
Timer() {
        }


/**
 **  start_timer
 **
*/
        public void start_timer (String timer_name) {
         int i;

TS1 = java.lang.System.currentTimeMillis();
TS1_name = timer_name;
} // end start_timer


/**
 **  end_timer
 **
*/
        public void end_timer (String timer_name) {
         int i;

if (timer_name.equals(TS1_name)) {
        TS2 = java.lang.System.currentTimeMillis();
        TS2_name = timer_name;
        if (GUI.Debug == -1)
              GUI.P((-1),"end_timer","Time for "+TS1_name+" is "+(TS2-TS1)+"
milliseconds");
        }
else
        if (GUI.Debug == -1)
              GUI.WARNING((-1),"end_timer","Wrong name: TS="+TS1_name+",
TS2="+TS2_name);
} // end end_timer

} // end class Timer
```

```
// This is ColorObj.java     by Rocky Nevin

import java.awt.*;

class ColorObj extends Object {
static Color VeryLightGrey, LightGrey, Grey, DarkGrey, VeryDarkGrey,
VeryLightRed, LightRed, Red, DarkRed, Crimson, VeryLightGreen, LightGreen,
Green, DarkGreen, Blue, VeryLightBlue, LightBlue, DarkBlue, DarkYellow, Yellow,
LightYellow, VeryLightYellow;

static int MAX_BACKGROUND_COLORS = 4;
static int background_color_index=0;
static Color background_color_array[] = new Color[MAX_BACKGROUND_COLORS];

// Color TempColor;
static Color RedArray[]    = new Color[16];
static Color PurpleArray[]= new Color[16];
static Color BlueArray[]   = new Color[16];
static Color GreenArray[]  = new Color[16];
static Color BlackArray[]  = new Color[16];


        public void ColorObj () {
                init();
                }

        public void init () { // ColorObj.init
                int i;
        background_color_array[0] = new Color(0x444444);
        background_color_array[1] = new Color(0x112211);
        background_color_array[2] = new Color(0x111122);
        background_color_array[3] = new Color(0xffffff);
        VeryLightGrey = new Color(0xeeeeee);
        LightGrey = new Color(0xaaaaaa);
        Grey = new Color(0x777777);
        DarkGrey = new Color(0x555555);
        VeryDarkGrey = new Color(0x110a0a);
        DarkYellow = new Color(0x222200);
        Yellow = new Color(0xaaaa44);
        LightYellow = new Color(0xaaaa44);
        VeryLightYellow = new Color(0xffff88);
        VeryLightRed = new Color(0xffdddd);
        LightRed = new Color(0xff4444);
        Red = new Color(0xaa4444);
        DarkRed = new Color(0x663333);
        LightRed = new Color(0xff4444);
        Crimson = new Color(0xee2299);
        DarkGreen = new Color(0x336633);
        Green = new Color(0x44aa44);
        LightGreen = new Color(0x77aa77);
        VeryLightGreen = new Color(0xddffdd);
        Blue = new Color(0x4444aa);
        LightBlue = new Color(0x777799);
        VeryLightBlue = new Color(0xddddff);
        DarkBlue = new Color(0x333366);

/*****************************************************************
                for (i=0; i<16; i++)
                        RedArray[i] = new Color((i)*0x001111 + 0xcc0000);
                for (i=0; i<16; i++)
                        PurpleArray[i] = new Color((i)*0x110011 + 0xcc0044);
                for (i=0; i<16; i++)
                        BlueArray[i] = new Color((i)*0x111100 + 0x0000cc);
                for (i=0; i<16; i++)
                        GreenArray[i] = new Color((i)*0x110011 + 0x00cc00);
                for (i=0; i<16; i++)
                        BlackArray[i] = new Color((i) * 0x111111);
*************************************************************/

// IF DEMO MODE, for clarity

                for (i=0; i<16; i++)
                        RedArray[i] = new Color((i)*0x110000 + (i/2)*0x001111);
                for (i=0; i<16; i++)
                        PurpleArray[i] = new Color((i)*0x110011 + (i/2)*0x001100);
                for (i=0; i<16; i++)
                        BlueArray[i] = new Color((i)*0x000011 + (i/2)*0x111100);
                for (i=0; i<16; i++)
                        GreenArray[i] = new Color((i)*0x001100 + (i/2)*0x110011);
                for (i=0; i<16; i++)
                        BlackArray[i] = new Color((i) * 0x111111);
// BlackArray has 16 elements, 0th is black, 10th is white, we use first 10,
index=(10-mag)

                } // end ColorObj.init

/*
** set_color_from_TS
*/
public void set_color_from_TS (Graphics graphics, Node node) {
        int index=20;
        double Tdiff, val;
        Color color=Color.yellow;

        if (node==null)
                return;
        else
                Tdiff = GUI.current_TS - node.TS;

if (node.isForm) {
        graphics.setColor(Color.white);
        return;
        }

        if (Tdiff <  2000) {
              val = 0.1*(4000.0-Tdiff)/4000.0 * node.delta_mag;
              index = (int)((val + 1.0) * 10); // range is 0 to +infinity

        switch (index) {
              case 0: color = BlueArray[15]; break;
              case 1: color = BlueArray[14]; break;
              case 2: color = BlueArray[13]; break;
              case 3: color = BlueArray[12]; break;
              case 4: color = BlueArray[11]; break;
              case 5: color = BlueArray[10]; break;
              case 6: color = BlueArray[9]; break;
              case 7: color = BlueArray[8]; break;
              case 8: color = BlueArray[7]; break;
              case 9: color = BlueArray[6]; break;
              case 10: color = RedArray[8]; break;
              case 11: color = RedArray[9]; break;
              case 12: color = RedArray[9]; break;
              case 13: color = RedArray[10]; break;
              case 14: color = RedArray[10]; break;
              case 15: color = RedArray[10]; break;
              case 16: color = RedArray[11]; break;
              case 17: color = RedArray[11]; break;
              case 18: color = RedArray[12]; break;
              case 19: color = RedArray[13]; break;
              case 20: color = RedArray[15]; break;
              default: color = Crimson; break;
              }
        graphics.setColor(color);
        }

        else  // neither Tdiff < 4000 nor val!=10
              set_color_from_mag(graphics, node);
        return;
} // end set_color_from_TS


/*
** set_color_for_relations
*/
public void set_color_for_relations (Graphics graphics, Node parent, Node child)
{
        int index=20;
        double Tdiff, val;
        Color color=Color.yellow;
        if (child==null)
                color = Crimson;
        else if (parent==null)
                color = Red;
        else {
        if (background_color_index < MAX_BACKGROUND_COLORS-1)
                index = (int)(4+Math.floor(child.mag));
        else
                index = (int)(Math.floor(1.5*(10-child.mag)));

        if (index>=0 && index<16)
                color = BlackArray[index];
        else
        if (index<0)
                color = Red;
        else
        if (index>16)
                color = Blue;

/*****************************************************
        switch (index) {
                case 0: color = BlackArray[3]; break;
                case 1: color = BlackArray[3]; break;
                case 2: color = BlackArray[3]; break;
                case 3: color = BlackArray[2]; break;
                case 4: color = BlackArray[2]; break;
                case 5: color = BlackArray[2]; break;
                case 6: color = BlackArray[1]; break;
                case 7: color = BlackArray[1]; break;
                case 8: color = BlackArray[1]; break;
                case 9: color = BlackArray[0]; break;
                case 10: color = BlackArray[0]; break;
                case 11: color = BlackArray[15]; break;
                case 12: color = BlackArray[15]; break;
                case 13: color = BlackArray[15]; break;
                case 14: color = BlackArray[15]; break;
                case 15: color = BlackArray[15]; break;
                case 16: color = BlackArray[15]; break;
                default: color = Crimson; break;
                }
*************************************************/
        }

graphics.setColor(color);

        return;
} // end set_color_for_relations

/*
** set_color_from_mag
```

```
*/
        public void set_color_from_mag (Graphics graphics, Node node) {
                double mag=0;

                if (node.isMarked) {
                        graphics.setColor((Color.cyan).darker());
                } else
                if (GUI.showBoundaries) {
                        switch ((int)node.Tdist) {
                        case 1: graphics.setColor(DarkGreen); break;
                        case 2: graphics.setColor(LightGreen); break;
                        case 3: graphics.setColor(VeryLightGreen); break;
                        case 4: graphics.setColor(LightRed); break;
                        case 5: graphics.setColor(DarkRed); break;
                        default: graphics.setColor(Crimson); break; // shouldn't
be here
                        }
                } else {

                if (node != null)
                        mag = node.mag;
// FIRST SET OF COLORS IS FOR DARK BACKGROUNDS
        if (background_color_index < MAX_BACKGROUND_COLORS-1) {
                if (node.isAN) {
                if (mag >= 0 && mag <= Node.MAX_MAG) {
                        if (mag <= Node.MED_MAG)
                                graphics.setColor(DarkGreen);
                        else if (mag <= Node.BIG_MAG)
                                graphics.setColor(Green);
                        else if (mag <= Node.MAX_MAG-0.1)
                                graphics.setColor(LightGreen);
                        else  if (mag >= Node.MAX_MAG-0.1)
                                graphics.setColor(VeryLightGreen); // maximum
brightness
                        else
                        graphics.setColor(Crimson); // shouldn't be here
                        }
                        }
                else
                if (node.isCN) {
                if (mag >= 0 && mag <= Node.MAX_MAG) {
                        if (mag <= Node.MED_MAG)
                                graphics.setColor(DarkBlue);
                        else if (mag <= Node.BIG_MAG)
                                graphics.setColor(Blue);
                        else if (mag <= Node.MAX_MAG-0.1)
                                graphics.setColor(LightBlue);
                        else  if (mag >= Node.MAX_MAG-0.1)
                                graphics.setColor(VeryLightBlue); // maximum
brightness
```

```
                    else
                    graphics.setColor(Crimson); // shouldn't be here
                    }
                    }
              else
              if (node.isDN) {
              if (mag >= 0 && mag <= Node.MAX_MAG) {
                    if (mag <= Node.MED_MAG)
                          graphics.setColor(DarkYellow);
                    else if (mag <= Node.BIG_MAG)
                          graphics.setColor(Yellow);
                    else if (mag <= Node.MAX_MAG-0.5)
                          graphics.setColor(LightYellow);
                    else  if (mag >= Node.MAX_MAG-0.5)
                          graphics.setColor(VeryLightYellow); // maximum
brightness
                    else
                    graphics.setColor(Crimson); // shouldn't be here
                    }
                    }

              else
              if (mag >= 0 && mag <= Node.MAX_MAG) {
                    if (mag <= Node.MED_MAG)
                          graphics.setColor(DarkGrey);
                    else if (mag <= Node.BIG_MAG)
                          graphics.setColor(Grey);
                    else if (mag <= Node.MAX_MAG-0.1)
                          graphics.setColor(LightGrey);
                    else  if (mag >= Node.MAX_MAG-0.1)
                          graphics.setColor(VeryLightGrey); // maximum
brightness
                    else
                    graphics.setColor(Crimson); // shouldn't be here
                    }


              } else {
// SECOND SET OF COLORS IS FOR LIGHT BACKGROUND
              if (node.isEvent) {
                    if (mag <= Node.MED_MAG)
                          graphics.setColor(VeryLightGrey);
                    else if (mag <= Node.BIG_MAG)
                          graphics.setColor(LightGrey);
                    else if (mag <= Node.MAX_MAG-0.1)
                          graphics.setColor(Grey);
                    else  if (mag >= Node.MAX_MAG-0.1)
                          graphics.setColor(DarkGrey); // maximum contrast
                    else
                    graphics.setColor(Crimson); // shouldn't be here
```

```
                        }
                else {
                if (mag >= 0 && mag <= Node.MAX_MAG)
                        graphics.setColor(BlackArray[(int)Math.floor(1.5*(10-
mag))]);
                else
                        graphics.setColor(Color.black); // maximum brightness
                }
                }
        }
                        //graphics.setColor(BlackArray[(int)Math.floor(mag)]);
        return;
        } // end set_color_from_mag


/*
** set_link_color_from_mag
* ***************************************************************************
        public void set_link_color_from_mag (Graphics graphics, Node node) {
                double mag=0;
                if (node != null)
                        mag = node.mag;
                if (mag > 0 && mag < 10)
                        graphics.setColor(BlueArray[(int)Math.floor(mag)]);
                else
                        graphics.setColor(BlueArray[(int)9]);//default maximum
darkness
        return;
        } // end set_link_color_from_mag
*****************************************************************/
 }      // End of class ColorObj
```

```
// This is Link.java     by Rocky Nevin

import java.lang.*;

/*
 * This is Link.java     by Rocky Nevin
 * @version      0.4, 3/12/98
 *
 */

public class Link extends Object {
static public final double DEFAULT_CS = 1.0;
static public final double MED_CS = 1.0;
static public final double MIN_CS = 0.1;
static public final double MAX_CS = 1.0;
static public final double DELTA_CS = 0.5;
String Name="";
String Type="";
String Desc_R="";
String Desc_L="";
double CS_R=DEFAULT_CS, CS_L=DEFAULT_CS;
Node Node;
Node CNode=null;
Node NodeL, NodeR;
Link NextLink;
VRObj VR_L, VR_R;
boolean should_we_pos_L=true, should_we_pos_R=true;
boolean isPolarized=false; // Bi-directional Link is polarized

/**
 **    Link   CONSTRUCTORS
 **
*/

        public Link() {
        VR_L = new VRObj();
        VR_R = new VRObj();
//  RECURSES   this.Node = new Node();
        }

/**
 **  set_CS
 **
*/
public void set_CS (double new_CS) {
this.CS_L = new_CS;
this.CS_R = new_CS;

if (this.CS_R > 5)
        this.CS_R = (5+this.CS_R)/2;
```

```
if (this.CS_L > 5)
        this.CS_L = (5+this.CS_L)/2;
if (this.CS_R < DELTA_CS)
        this.CS_R = DELTA_CS;
if (this.CS_L < DELTA_CS)
        this.CS_L = DELTA_CS;

} // end set_CS

/**
 **  more_CS
 **
*/
public void more_CS () {
        this.CS_R += DELTA_CS;
        this.CS_L += DELTA_CS;
if (this.CS_R > 5)
        this.CS_R = (5+this.CS_R)/2;
if (this.CS_L > 5)
        this.CS_L = (5+this.CS_L)/2;

} // end more_CS

/**
 **  less_CS
 **
*/
public void less_CS () {
this.CS_R -= DELTA_CS;
this.CS_L -= DELTA_CS;
if (this.CS_R < DELTA_CS)
        this.CS_R = DELTA_CS;
if (this.CS_L < DELTA_CS)
        this.CS_L = DELTA_CS;

} // end less_CS



/**
 **  addCNode
 **
*/
public void addCNode (Node CNode) { // CSs
int i, size;
Node child;

//System.out.println("Link.addCNode: adding "+CNode.Name+" to this Link and vice
versa");
this.CNode = CNode;
```

```
if (! CNode.ContextLinks.contains(this)) {
       CNode.ContextLinks.addElement(this);
//     System.out.println("Link.addCNode: NEW: "+CNode.Name+" modulates "+
CNode.ContextLinks.size() +" links.");
       CNode.isCN = true;
       }
//else
//     System.out.println("Link.addCNode: ALREADY ADDED: "+CNode.Name+"
modulates "+ CNode.ContextLinks.size() +" links.");
} // end addCNode


/**
 **  setLinks_should_we_pos   in single-caller version positioning 6/22/99
 **
*/
public void setLinks_should_we_pos (Node node, boolean value) {
int which=0;

if (node == null) {
       System.out.println("getNodesVRparms(): ERROR, passed node is null");
       return;
       }

// determine if we should get Left or Right
if (node == this.NodeL) // use Left parms
       should_we_pos_L = value;
else
if (node == this.NodeR) // use Right parms
       should_we_pos_R = value;

return;
} // end setLinks_should_we_pos


/**
 **  should_we_pos   in single-caller version positioning 6/22/99
 **
*/
public boolean should_we_pos (Node node) {
int which=0;

if (node == null) {
       System.out.println("getNodesVRparms(): ERROR, passed node is null");
       return(false);
       }

// determine if we should get Left or Right
if (node == this.NodeL) // use Left parms
```

```
        return(should_we_pos_L);
else
if (node == this.NodeR) // use Right parms
        return(should_we_pos_R);

return(true); // have to have a default
} // end should_we_pos


/**
 **  setNodesVRparms   get VR parms in link from passed node
 **
*/
public void setNodesVRparms (Node node) {
int which=0;

if (node == null) {
        System.out.println("getNodesVRparms(): ERROR, passed node is null");
        return;
        }

// determine if we should get Left or Right
if (node == this.NodeL) // get Left parms
        which = 1;
else
if (node == this.NodeR) // get Right parms
        which = 2;


// now make assignments, make Left negative of Right, since offsets are simple
switch (which) {
        case 1:  // Set link's Left X to node's X, link's Right X to -nodes's X
                node.X = this.VR_L.X;
                node.Y = this.VR_L.Y;
                break;
        case 2: // Set link's Right X to node's X, link's Left X to -nodes's X
                node.X = this.VR_R.X;
                node.Y = this.VR_R.Y;
                break;
        default:
                System.out.println("getNodesVRparms(): ERROR, which = "+which);
                break;
        }
return;
} // end setNodesVRparms


/**
 **  setLinksVRparms   set VR parms in link from passed node
 **
```

```
*/
public void setLinksVRparms (Node node) {
int which=0;

if (node == null) {
        System.out.println("setLinksVRparms(): ERROR, passed node is null");
        return;
        }

// determine if we should set Left or Right
if (node == this.NodeL) // set Left parms
        which = 1;
else
if (node == this.NodeR) // set Right parms
        which = 2;


// now make assignments, make Left negative of Right, since offsets are simple
switch (which) {
        case 1:  // Set link's Left X to node's X, link's Right X to -nodes's X
                this.VR_L.X = node.X;
                this.VR_L.Y = node.Y;
                this.VR_R.X = -this.VR_L.X;
                this.VR_R.Y = -this.VR_L.Y;
                break;
        case 2: // Set link's Right X to node's X, link's Left X to -nodes's X
                this.VR_R.X = node.X;
                this.VR_R.Y = node.Y;
                this.VR_L.X = -this.VR_R.X;
                this.VR_L.Y = -this.VR_R.Y;
                break;
        default:
                System.out.println("setLinksVRparms(): ERROR, which = "+which);
                break;
        }
return;
} // end setLinksVRparms


/**
 **  dumpLink
 **
*/
public void dumpLink () {
String nodeLname="null", nodeRname="null";

if (NodeL != null)
        nodeLname = NodeL.Name;
if (NodeR != null)
        nodeRname = NodeR.Name;
System.out.println("Link.dumpLink: '"+nodeLname+"' ----> '"+nodeRname+"'
X="+this.VR_R.X);
System.out.println("Link.dumpLink: '"+nodeRname+"' ----> '"+nodeLname+"'
X="+this.VR_L.X);

} // end dumpLink


public class VRObj extends Object {
        // some elements for data, stubbed now
        double X, Y, Z; // relative positions in VR space, typically
                                // positions offsets from another node
        double size_x, size_y, size_z; // sizes
//      VRShape Shape; // data and methods to render semi-realistically, for VR

        public VRObj() {
                ;
        }
} // end internal class VRObj



}       // End of class Link
```

```
// This is Mode.java     by Rocky Nevin


class Mode extends Object {
    String Name="AND";
    String spread_mode="distal";
    String lines_mode="all";
    String position_mode="Rel'";
    String render_mode="VR";
    String line_mode="radial";
    boolean do_potentiation = false;

    public Mode() {
        super();
        } // end Mode creator

/**
 **  set_mode_buttons
 */
void set_mode_buttons() {
        //GUI.button_spread.setLabel("Spread "+spread_mode);
        GUI.button_render.setLabel("Render "+render_mode);
        GUI.button_lines.setLabel("Lines "+lines_mode);
        GUI.button_position.setLabel("Pos'n "+position_mode);
        GUI.button_potentiation.setLabel("Pot "+do_potentiation);
        if (do_potentiation)
                GUI.button_potentiation.setBackground(ColorObj.LightRed);
        else
                GUI.button_potentiation.setBackground(ColorObj.LightGrey);
        GUI.button_drawText.setLabel("Text "+GUI.drawText);
        if (GUI.drawText)
                GUI.button_drawText.setBackground(ColorObj.LightRed);
        else
                GUI.button_drawText.setBackground(ColorObj.LightGrey);
} // end set_mode_buttons


/**
 **  toggle_draw_text
 **
*/
public void toggle_draw_text () {
        GUI.drawText = !GUI.drawText;
        GUI.P(1,"toggle_draw_text", " GUI.drawText = "+GUI.drawText);
        set_mode_buttons();
        }

/**
 *   method set_spread_mode
 */
```

```
void set_spread_mode (String str) {
        spread_mode = str;
GUI.P(1,"Mode.set_spread_mode","Mode "+str);
set_mode_buttons();
} // end set_spread_mode

/**
 *   method set_render_mode
 */
void set_render_mode (String str) {
        render_mode = str;
GUI.P(0,"Mode.set_render_mode","Mode "+str);
set_mode_buttons();
} // end set_render_mode

/**
 *   method set_position_mode
 */
void set_position_mode (String str) {
        position_mode = str;
GUI.P(1,"ModMode.set_position_mode","Mode "+str);
set_mode_buttons();
} // end set_position_mode


/**
 *   method toggle_lines_mode
 */
void toggle_lines_mode () { // radial, distal, proximal
GUI.P(1,"word_selector","lines_mode = "+lines_mode);
if (lines_mode.equals("radial")) {
        lines_mode = "distal";
        }
else if (lines_mode.equals("all"))
        lines_mode = "none";
else if (lines_mode.equals("none"))
        lines_mode = "local";
else if (lines_mode.equals("local"))
        lines_mode = "all";
set_mode_buttons();
} // end toggle_lines_mode

/**
 *   method toggle_spread_mode
 */
void toggle_spread_mode () { // radial, distal, proximal
GUI.P(1,"word_selector","spread_mode = "+spread_mode);
if (spread_mode.equals("radial")) {
        spread_mode = "distal";
        }
```

```
else if (spread_mode.equals("distal"))
        spread_mode = "proximal";
else if (spread_mode.equals("proximal"))
        spread_mode = "radial";
set_mode_buttons();
} // end toggle_spread_mode

/**
 *   method toggle_position_mode
 */
void toggle_position_mode () { // relations, levels
if (position_mode.equals("Rel'")) {
        position_mode = "Grid";
        Force.calcPOVPressure=true;
        Force.calcParentPressure=false;
System.out.println("toggle_position_mode: setting position_mode to
"+position_mode);
        }
else if (position_mode.equals("Grid")) {
        position_mode = "Lvls";
        Force.calcPOVPressure=true;
        Force.calcParentPressure=false;
System.out.println("toggle_position_mode: setting position_mode to
"+position_mode);
        }
else if (position_mode.equals("Lvls")) {
        position_mode = "Rel'";
        Force.calcPOVPressure=false;
        Force.calcParentPressure=true;
System.out.println("toggle_position_mode: setting position_mode to
"+position_mode);
        }
GUI.button_presParent.setLabel("Parent="+Force.calcParentPressure);
if (Force.calcParentPressure)
        GUI.button_presParent.setBackground(ColorObj.LightRed);
else
        GUI.button_presParent.setBackground(ColorObj.LightGrey);
GUI.button_presPOV.setLabel("POV="+Force.calcPOVPressure);
if (Force.calcPOVPressure)
        GUI.button_presPOV.setBackground(ColorObj.LightRed);
else
        GUI.button_presPOV.setBackground(ColorObj.LightGrey);
set_mode_buttons();
} // end toggle_position_mode

/**
 *   method toggle_spread_mode
 */
void toggle_line_mode () { //
if (line_mode.equals("radial"))
```

```
        line_mode = "distal";
else if (line_mode.equals("distal"))
        line_mode = "proximal";
else if (line_mode.equals("proximal"))
        line_mode = "radial";
set_mode_buttons();
} // end toggle_line_mode

/**
/**
 *   method toggle_render_mode
 */
void toggle_render_mode () {
System.out.println("Render_mode was = "+render_mode);
if (render_mode.equals("VR"))
        render_mode = "Rel'";
else if (render_mode.equals("Rel'"))
        render_mode = "VR";
set_mode_buttons();
System.out.println("Render_mode is = "+render_mode);
} // end toggle_render_mode

/**
 *   method toggle_potentiation_mode
 */
void toggle_do_potentiation () {
do_potentiation = !do_potentiation;
set_mode_buttons();
} // end toggle_do_potentiation

} // end Object Mode
```

```
// This is Node.java     by Rocky Nevin

import java.lang.*;
import java.util.*;
import java.awt.*;

/*
 * This is Node.java     by Rocky Nevin
 * @version     0.4, 3/12/98
 * by Rocky Nevin
 */

// link() now uses a Vector of Link objects
// position_children() returns if TS<=G.Ccurrent_TS and wiggles Root node
// 6/29/99 added Link.CS_R&L, set other node.CS to it in getNodeAtLink()

public class Node extends Object {

static final int MAX_CHILDREN = 90;
static final int MAX_TEXT_DATA_LINES = 120;
static final int MAX_NORMALIZED_MAG = 10;
static public final double DELTA_MAG = 1.0;
static public final double MIN_MAG = 0.2;
static public final double SMALL_MAG = 1.0;
static public final double MED_MAG = 4.0;
static public final double BIG_MAG = 7.0;
static public final double MAX_MAG = 10.0;
static public final double EMPHASIZED_MAG = 15.0;
static public final double HYPER_MAG = 20.0;
static public final double BARELY_VISIBLE_MAG = SMALL_MAG + 0.1;
static public final double BARELY_INVISIBLE_MAG = SMALL_MAG - 0.1;
static public final double DEFAULT_MAG = MED_MAG-0.5;
boolean Debug=false;
boolean StopSpread=false;
boolean isFrozen=false; // Temporary freezing of position
boolean isPolarized=true; // Bi-directional Link is polarized
boolean isSelected=false;
boolean isBB=false;
boolean isMarked=false;
boolean isFile=false;  //
boolean isDirectory=false;  //
boolean isPN=false;  // is Potentiation node
boolean isCN=false;  // is Context node
boolean isAN=false;  // is Abstract node
boolean isON=false;  // is Abstract node
boolean isDN=false;  // is Data node
boolean isURL=false; // is URL node
boolean isPOV=false;
boolean isEvent=false;
boolean isForm=false;
boolean isLayer=false;
boolean there_is_data=false;
boolean data_access_failed=false;
boolean there_should_be_data=false;
double dist = 0;
double old_dist = 0;
int Tdist = 0;
int AN_level = 0;
int ID = 0;  // Used to track temporary actions on any node
int ChildNum = 0;  // child number of this, counted by its parent (where
parent.dist=this.dist-1)
int ChildCount = 0;  // the number of children of this (where
child.dist=this.dist+1)
int BigChildCount = 0;  // How many emphasized children there are
int TransitionCount=0;
String Type="";
String Name="";
String Desc="";
String Data[];
// double TinyScale=1.0;
double ThetaOffset = 0;
// long LongData;
long created_TS = 0; // A Time Stamp to record when we last did something to
ourself,
long TS = 0; // A Time Stamp to record when we last did something to ourself,
long drawnTS = 0; // A Time Stamp to record when we last did something to
ourself,
Vector Links;
Vector ContextLinks;
int child_vec[]; // allows us to order fn's called on children
double px=0, py=0;
double pxPOV=0, pyPOV=0;
double pxNeighbors=0, pyNeighbors=0;
double pxParent=0, pyParent=0;
Color TempColor;
static int potentiation=1;
long potentiation_TS = -1; // A Time Stamp to record when we last were
potentiated
double potentiation_mag = 1;
double importance = 1;
// double CS = Link.DEFAULT_CS;
double mag = DEFAULT_MAG;
double min_mag = 0;
long   magTS;
double old_mag = 0.1;
double delta_mag = 1.0;
double X=0, Y=0, Z=0;           // offsets from parent in VRmode in data
corrdinates
double size_X=6, size_Y=15, dZ=1; //
double x=0, y=0, z=0;           // temp positions for POVs in data corrdinates
double size_x=1, size_y=1, dz=1;  // permanent sizes for objects in data
coordinates
double mag1=1.0, mag2=1.0, mag3=1.0;
long   mag1TS, mag2TS, mag3TS;
// Node VR_node;
boolean VRyes;
// static Node Myself;
// double DoubleData;
// DataObj dataobj;  // I'm storing data directly into nodes now (5/99)
// Vector GlobalVec;
// static int total_nodes_created = 0;
/**
   DS.spreadback_from(target_node, POV)
   is a mechanism for finding the shortest routefrom a POV (or any node) to
   any other node. It spreads one level at a time, thus invoked for n-times
   until all the return values indicate no more nodes.
   It uses TS each invocation, passes desired level to return form.
   The invocation which finds the target returns a code and the caller
   modifies the mag to that invocation, spreading backwards directly to the POV
   */

/**
 **   Node CONSTRUCTORS   (constructors)
 **
*/

//
        public Node() {
        init();
         assign();
         }

        public Node (String Name) {
        init(); this.Name = Name; //createMyself();
         assign();
        }
        public Node (String Name, String Type) {
        init(); this.Name = Name; this.setType(Type); //createMyself();
         assign();
        }
        public Node (String Name, String Type, String Desc) {
        init(); this.Name = Name; this.setType(Type); this.Desc = Desc;
 //createMyself();
         assign();
        }
        public Node (String Name, String Type, String Desc, String Data) {
        init(); this.Name = Name; this.setType(Type); this.Desc = Desc;
 //createMyself();
        this.Data[0] = Data;
         assign();
        }

        public Node (String Name, String Type, String Desc,
                        int x, int y) {
        init(); this.Name = Name; this.setType(Type); this.Desc = Desc;
         //this.X = x; this.Y = y; //createMyself();
        this.setX(x,y);
         assign();
        }

        public Node (String Name, String Type, String Desc, String Data,
                        int x, int y) {
        init(); this.Name = Name; this.setType(Type); this.Desc = Desc;
         //this.X = x; this.Y = y; //createMyself();
        this.setX(x,y);
        this.Data[0] = Data;
         assign();
        }

        public Node (String Name, String Type, String Desc,
                        int x, int y, int size_x, int size_y) {
        init(); this.Name = Name; this.setType(Type); this.Desc = Desc;
        //this.X = x; this.Y = y;
        this.size_X = size_x; this.size_Y = size_y; //createMyself();
        this.setX(x,y);
         assign();
        }

        public Node (String Name, String Type, String Desc, String Data,
                        int x, int y, int size_x, int size_y) {
        init(); this.Name = Name; this.setType(Type); this.Desc = Desc;
        //this.X = x; this.Y = y;
        this.size_X = size_x; this.size_Y = size_y; //createMyself();
        this.setX(x,y);
        this.Data[0] = Data;
         assign();
        }

/**
 **  Node.setX   another place to do all setting of X variables
 **
*/
        public void setX (double X, double Y) {
        double oldX=0, oldY=0;

        this.setX("-?-", X, Y);
        return;
} // end Node.setX
```

```
/**
 **  Node.setX   another place to do all setting of X variables
 **
*/
        public void setX (String CallingFnName, double X, double Y) {
        double oldX=0, oldY=0;

        oldX = this.X;
        oldY = this.Y;
        this.X = X;
        this.Y = Y;

// HERE WE CAN MONITOR PROBLEM AREAS
/****************************************
        if (
                (this.Name.equalsIgnoreCase("TL"))
                || (this.Name.equalsIgnoreCase("Today"))
                || (this.Name.equalsIgnoreCase("2pm:Today:"))
                || (this.Name.equalsIgnoreCase("2pm:Tomorrow:"))
                || (this.Name.equalsIgnoreCase("2pm:Yesterday:"))
                || (this.Name.equalsIgnoreCase("cal"))
                || (this.Name.equalsIgnoreCase("Root"))
                )
        if (oldX==X && oldY==Y)
                System.out.println("Node.setX : "+CallingFnName+
                        " '"+this.Name+"'.Xvar was ("+oldX+","+oldY+") and is
unchanged");
        else
                System.out.println("Node.setX : "+CallingFnName+
                        " '"+this.Name+"'.Xvar was ("+oldX+","+oldY+") and is
changed to -> ("+X+","+Y+")");
****************************************/
        return;
} // end Node.setX


/**
 **  set_pot  set the potentiation_TS to the argument
 **
*/
public void set_pot () {
        this.set_pot(GUI.current_TS);
}

public void set_pot (long pot_TS) {
        this.potentiation_TS = pot_TS;
        // System.out.println(" set_pot("+this.Name+")" );
} // end set_pot
```

```
/*
 * assign   set Node vars, and set Link's vars from Node's
 */
public void assign () {
Node tnode=null;
double dx=50*(GUI.random()-0.5), dy=50*(GUI.random()-0.5);

Link link;
        this.x = this.X + dx;
        this.y = this.Y + dy;
        this.size_x = this.size_X;
        this.size_y = this.size_Y;
        DataSea.add_to_node_vec(this); // this will initialize node_vec if needed
} // end assign


/**
 **  setType
 **
*/
public void setType (String type) {
int i, size;
Node tn;


        this.Type = type;

        if (type.equalsIgnoreCase("BB"))
                this.isBB = true;
        if (type.equalsIgnoreCase("PN"))
                this.isPN = true;
        if (type.equalsIgnoreCase("CN"))
                this.isCN = true;
        if (type.equalsIgnoreCase("AN"))
                this.isAN = true;
        if (type.equalsIgnoreCase("ON"))
                this.isON = true;
        if (type.equalsIgnoreCase("POV"))
                this.isPOV = true;
        if (type.equalsIgnoreCase("DN"))
                this.isDN = true;
        if (type.equalsIgnoreCase("Event"))
                this.isEvent = true;
        if (type.equalsIgnoreCase("Form"))
                this.isForm = true;
        if (type.equalsIgnoreCase("Layer"))
                this.isLayer = true;
        if (type.equalsIgnoreCase("URL")) {
                this.isURL = true;
                }

        this.Data[0] = "No Data available for node <"+this.Name+">";

} // end setType

/*
 * init   create a vector called this.Links
 */
public void init () {
Link link;
int i;
        this.Links = new Vector(10);
        this.ContextLinks = new Vector(1);
        this.child_vec = new int[MAX_CHILDREN+1];
        this.created_TS = GUI.current_TS;
        this.set_TS();
        this.Data = new String[MAX_TEXT_DATA_LINES];
} // end init

/**
 **  setLinks_should_we_pos
 **
*/
        public void setLinks_should_we_pos (Node node, boolean value) {
        Link link;
        if (node == null) {
                System.out.println("Node.setLinks_should_we_pos(): ERROR, node is
null");
                return;
                }
        link = node.getLinkTo(node);
        if (link == null) {
                System.out.println("Node.setLinks_should_we_pos(): ERROR, link is
null to "
                                +node.Name);
                if (GUI.animation_thread != null)
                        GUI.animation_thread.dumpStack();
                return;
                }
        link.setLinks_should_we_pos(node, value);
} // end setLinks_should_we_pos


/**
 **  setLinksVRparmsTo
 **
*/
        public void setLinksVRparmsTo (Node node) {
```

```
        Link link;
        if (node == null) {
                System.out.println("Node.setLinksVRparmsTo(): ERROR, node is
null");
                return;
                }
        link = node.getLinkTo(node);
        if (link == null) {
                System.out.println("Node.setLinksVRparmsTo(): ERROR, link is null
to "+node.Name);
                if (GUI.animation_thread != null)
                        GUI.animation_thread.dumpStack();
                return;
                }
        link.setLinksVRparms(node);
} // end setLinksVRparmsTo



/*
**
**/
public Node getParent (Node node){
Node tnode, saved_node=null;
int i, size;
if (node == null)
   return((Node)null);

size = node.Links.size();
for (i=0; i<size; i++) {
    tnode = node.getNodeAtLink(i);
    if (tnode==null) {
        break;
        }
    if (tnode.dist == (node.dist - 1)){
        saved_node = tnode;
        }
    }

return(saved_node);
} // end getParent



/**
 *   method checkSamePol  tell if links from parent->this and this->child have
the same polarization
 */
public boolean checkSamePol (Node parent, Node child) {
```

```
char parentPol, childPol;
boolean returned_boolean=false;

return(true); // SIMPLIFY 11/11/99

/************************************************************
if (!GUI.checkPolarization)
       return(true);

if (!parent.isPolarized)
       returned_boolean = true;
if (!this.isPolarized)
       returned_boolean = true;
if (!child.isPolarized)
       returned_boolean = true;


if (returned_boolean == false) {
parentPol = parent.getPol(this);
childPol = this.getPol(child);

if (parentPol == childPol)
       returned_boolean = true;
else
       returned_boolean = false;
}

if (GUI.Debug == 1)
 System.out.println("checkSamePol(): "+returned_boolean+" between
"+parent.Name+"("+parent.Type+") -> "+this.Name+"("+parent.Type+") ->
"+child.Name+"("+parent.Type+")");

return(returned_boolean);
************************************************************/
} // end checkSamePol



/**
 **  hasSmallerDistThan   Returns a sibling which is of Type 'type', else null
 **
*/
public boolean hasSmallerDistThan (Node other_node) {
boolean ret_val;

if (this.dist < other_node.dist - 0.00001) // handles java imprecision
       ret_val = true;
else
       ret_val = false;
```

```
return(ret_val);
} // end hasSmallerDistThan



/**

/**
 **  hasLargerDistThan    Returns a sibling which is of Type 'type', else null
 **
*/
public boolean hasLargerDistThan (Node other_node) {
boolean ret_val;

if (this.dist > other_node.dist + 0.00001) // handles java imprecision
        ret_val = true;
else
        ret_val = false;

return(ret_val);
} // end hasLargerDistThan



/**
 **  hasSiblingOfType    Returns a sibling which is of Type 'type', else null
 **
*/
public Node hasSiblingOfType (String type) {
int i, size;
Node child;

size = this.Links.size();
for (i=0; i<size; i++) {
        child = (Node)(this.getNodeAtLink(i));
        if (child.Type.equalsIgnoreCase(type))
                return(child);
        }

return((Node)null);
} // end hasSiblingOfType


/**
 **  goesUpstreamTo
 **
*/
public boolean goesUpstreamTo (Node node) {
int i, size;
Node child;
```

```
if ('-' == this.getPol(node))
       return(true);
else
       return(false);

} // end goesUpstreamTo



/**
/**
 **  goesDownstreamTo
 **
*/
public boolean goesDownstreamTo (Node node) {
int i, size;
Node child;

if ('+' == this.getPol(node))
       return(true);
else
       return(false);

} // end goesDownstreamTo



/**
 *   method getPol        get the polarization between this and node
 */
public char getPol (Node node) {
char returned_char='-';


if (node==null) {
       GUI.WARNING(0, "Node.getNodeAtLink", "node is null");
       return('x');
       }

Link link = this.getLinkTo(node);
if (link==null) {
       GUI.WARNING(0, "Node.getNodeAtLink", "link between <"+this.Name
               +"> and <"+node.Name+"> is null");
       return('x');
       }
if (node == link.NodeR)
       returned_char = '+';
else
       returned_char = '-';
```

```
return(returned_char);
} // end getPol


/**
 *   method getAN_connected_to_DN         UN-NECESSARY?
 */
        public Node getAN_connected_to_DN (Node DN) {
        int i, j, size;
        Node tAN, tDN;

         size = this.Links.size();
        for (i=0; i<size; i++) {
                tAN = this.getNodeAtLink(i);
                if (tAN == null)
                        return((Node)null);
                j = 0; // set to run through DN's
                if (tAN.isAN) {
                        while (null != (tDN = tAN.getDN(j++)))
                        {
                        if (tDN == DN)
                                return(tAN); // tAN is connected to arg DN
                        else
                                ; // Keep trying to find the given DN connected to
tAN
                        }

                }
        }

         return((Node)null);
 } // end getAN_connected_to_DN

 /**
 *   method getDN_connected_to_AN         UN-NECESSARY?
 */
        public Node getDN_connected_to_AN (Node AN) {
        int i, j, size;
        Node tDN, tAN;

         size = this.Links.size();
        for (i=0; i<size; i++) {
                tDN = this.getNodeAtLink(i);
                if (tDN == null)
                        return((Node)null);
                j = 0; // set to run through AN's
                if (tDN.isDN) {
                        while (null != (tAN = tDN.getAN(j++)))
```

```
                                {
                                if (tAN == AN)
                                        return(tDN); // tDN is connected to arg AN
                                else
                                        ; // Keep trying to find the given AN connected to
tDN
                                }

                        }
                }

                return((Node)null);
        } // end getDN_connected_to_AN

        /**
         **  getParent
         **
        */
                public Node getParent () {
                int size=0, i;
                Node tnode;

                size = this.getChildCount();
                for (i=0; i<size; i++) {
                        tnode = this.getChild(i);
                        if (tnode.dist < this.dist)
                                return(tnode); // parent will have lower dist
                        }
                return((Node)null); // return null if we fall through
        } // end getParent


        /**
         **  getChild
         **
        */
                public Node getChild (int i) {
                return(this.getNodeAtLink(i));
        } // end getChild


        /**
         **  getChildCount
         **
        */
                public int getChildCount () {
                 return(this.Links.size());
        } // end getChildCount
```

```
/**
 *   method getAN_named        return a (remotely) linked AN, go dist of
'how_far' max.
 *                             Later allow return of distance info and such
 */
        public Node getAN_named (String name, int how_far) {
        int i, AN_counter=0, size;
        Node tnode=null, ret_node=null;

        if (how_far <= 0) // SINCE WE ARE RECURSIVE, CONSIDER THE END POINT
                return( (Node)null ); // FAILED TO FIND NODE


System.out.println("getAN_named(): looking for '"+name+"' from
this='"+this.Name+"'");

         size = this.Links.size();
        for (i=0; i<size; i++) {

                tnode = this.getNodeAtLink(i);
                System.out.print("getAN_named(): how_far="+how_far+",
tnode.Name='"+tnode.Name+"';" );
                if (tnode == null)
                        return((Node)null);
                if (tnode.dist >= this.dist) {
                System.out.print(" dist="+tnode.dist+"(>="+this.dist+");");
                if (tnode.Type.equals("AN")) {
                        System.out.print(" is  an AN.");
                        if (tnode.Name.equals(name)) {
                                System.out.println("; CORRECT NAME, returning.");
                                return(tnode);
                        }
                        else
                        System.out.print(" but is the wrong name, != '"+name+"'
looping....");
                }
                else
                System.out.print(" not an AN.");
                }
                else
                System.out.print(" dist="+tnode.dist+" !> "+this.dist);
System.out.println("");
        }

// HAVEN'T FOUND CORRECT AN, SO RE-DO LOOP ON CHILDREN AND RECURSE WITH how_far-
-

        for (i=0; i<size; i++) {
```

```
                    tnode = this.getNodeAtLink(i);
                    if (tnode == null)
                            return((Node)null);
                    if (tnode.dist > this.dist) {
                            System.out.println("Recursing.");
                            ret_node = tnode.getAN_named(name, (how_far-1));
                            if (ret_node != null)
                                    return(ret_node);
                    }
            }
    // FALL THROUGH
    return((Node)null);
    } // end getAN_named


     /**
     *   method getAN        return a directly linked AN, skip 'which' of them
     */
            public Node getAN (int which) {
            int i, AN_counter=0, size;
            Node tnode;

             size = this.Links.size();
            for (i=0; i<size; i++) {
                    tnode = this.getNodeAtLink(i);
                    if (tnode == null)
                            return((Node)null);
                    if (tnode.Type.equals("AN")) {
                            if (AN_counter<which) // if which==0, return first
                                    AN_counter++;
                            else
                                    return(tnode);
                    }
            }

             return((Node)null);
     } // end getAN

     /**
     *   method getDN        return a directly linked DN, skip 'which' of them
     */
            public Node getDN (int which) {
            int i, DN_counter=0, size;
            Node tnode;

             size = this.Links.size();
            for (i=0; i<size; i++) {
                    tnode = this.getNodeAtLink(i);
                    if (tnode == null)
                            return((Node)null);
```

```
                if (tnode.Type.equals("DN")) {
                        if (DN_counter<which) // if which==0, return first
                                DN_counter++;
                        else
                                return(tnode);
                }
        }

        return((Node)null);
} // end getDN

 /**
 *   method getLinkTo
 */
        public Link getLinkTo (Node node) {
        int i, size, which=-1;
        Node tnode;

         size = this.Links.size();
        which = this.isThereLinkTo(node);

        if (which != -1)
                return((Link)(this.Links.elementAt(which))); // got it

        return((Link)null); // fall-through, default, a failure
         } // end getLinkTo


 /**
 *   method getLink
 */
        public Link getLink (int i) {

         return((Link)(this.Links.elementAt(i)));
         }

/**
 *   method isThereLinkTo   Get the other node, not 'this'
 */
        public int isThereLinkTo (Node node) {
         Link tlink;
        int i, size;

         size = this.Links.size();
        for (i=0; i<size; i++) {
                tlink = (Link)(this.Links.elementAt(i));
                if ((node == tlink.NodeL) || (node == tlink.NodeR))
                        return(i);
        }
        return(-1);
        } // end isThereLinkTo


    /**
     *   method set_Desc   set the Desc in the correct link to other_node
     */
        public int set_Desc (Node other_node, String desc) {

        if (desc.equals(""))
                return(0);

        Link link = getLinkTo(other_node);
        if (link == null)
                return(-1);

        if (this == link.NodeR) {
                link.Desc_R = desc;
                GUI.P(0,"Node.set_Desc", "Set Desc('"+desc+"') into link_R of
"+this.Name+" -> "+other_node.Name);
                }
        else
        if (this == link.NodeL) {
                link.Desc_L = desc;
                GUI.P(0,"Node.set_Desc", "Set Desc('"+desc+"') into link_L of
"+this.Name+" -> "+other_node.Name);
                }
        else {
                GUI.ERROR(0,"Node.set_Desc", "Internal error, this is neither
NodeR or NodeL");
                return(-1);
                }

        return(0);
        } // end set_Desc


    /**
     *   method get_Desc   get the Desc in the correct link to other_node
     */
        public String get_Desc (Node other_node) {
        String desc=null;

        Link link = getLinkTo(other_node);
        if (link == null)
                return(desc); // null

        if (this == link.NodeR) {
                desc = link.Desc_R;
                GUI.P(0,"Node.get_Desc", "Got ("+desc+") from link_R of
"+this.Name+" -> "+other_node.Name);
                }
        else
        if (this == link.NodeL) {
                desc = link.Desc_L;
                GUI.P(0,"Node.get_Desc", "Got ("+desc+") from link_L of
"+this.Name+" -> "+other_node.Name);
                }
        else {
                GUI.ERROR(0,"Node.get_Desc", "Internal error, this is neither
NodeR or NodeL");
                return(desc); // null
                }

        return(desc);
        } // end get_Desc

/**
 *    method set_CS   Set the CS from this to the other node
        public double set_CS (Node other_node, double CS) {

        Link link = getLinkTo(other_node);
        if (link == null)
                return(0);

        if (this == link.NodeL) {
                if (this.Debug && GUI.Debug == 1)
                        System.out.println("set_CS():("+this.Name+")-
>("+other_node.Name
                                +"): CS "+link.CS_R+" -> "+CS);
                link.CS_R = CS;
                }
        else
        if (this == link.NodeR) {
                if (this.Debug && GUI.Debug == 1)
                        System.out.println("set_CS():("+this.Name+")-
>("+other_node.Name
                                +"): CS "+link.CS_L+" -> "+CS);
                link.CS_L = CS;
                }
        else
                GUI.ERROR(0,"Node.set_CS", "Internal error, this is neither NodeR
or NodeL");

        return(CS);
        } // end set_CS
 */

/**
```

```
 *   method set_CS_from_CNode   Set the CS from this to the other node
 */
public double set_CS_from_CNode (Node left_node, Node right_node, double CS) {
// CSs

Link link = left_node.getLinkTo(right_node);
if (link == null)
       return(0);

link.set_CS(CS);

return(CS);
} // end set_CS_from_CNode


/**
 *   method get_CS   Get the other node, not 'this'
 */
        public double get_CS (Node other_node) {

        if (other_node == null)
               return(Link.DEFAULT_CS);

        Link link = getLinkTo(other_node);
        if (link == null)
               return(0);

        if (this == link.NodeR) {
               return(link.CS_L);
               }
        else
        if (this == link.NodeL) {
               return(link.CS_R);
               }
        else
               GUI.ERROR(0,"Node.get_CS", "Internal error, this is neither NodeR
or NodeL");
        return(0);
        } // end get_CS

/**
 *   method getNodeAtLink   Get the other node, not 'this'
 */
        public Node getNodeAtLink (int i) {
         Link tlink=null;

        if (i >= this.Links.size()) {
        GUI.WARNING(0, "Node.getNodeAtLink",
               "FAILED, i("+i+")>= size of Links
vector("+this.Links.size()+")");
```

```
                return((Node)null);
                }
        tlink = (Link)(this.Links.elementAt(i));
        if (this == tlink.NodeL) {     // decide which node to return
                this.get_CS(tlink.NodeR); // store CS into NodeR, use soon
                return(tlink.NodeR);        //  return 'other' node
        }
        else    {
                this.get_CS(tlink.NodeL); // store CS into NodeL, use soon
                return(tlink.NodeL);        //  return 'other' node
        }
} // end getNodeAtLink


/**
 *   method getCNodeAtLink   Get the CN node
 */
        public Node getCNodeAtLink (int i) {  // CSs
        Link tlink=null;

        if (i >= this.Links.size()) {
        GUI.WARNING(0, "Node.getCNodeAtLink",
                "FAILED, i("+i+")>= size of Links
vector("+this.Links.size()+")");
                return((Node)null);
                }
        tlink = (Link)(this.Links.elementAt(i));

return(tlink.CNode);
} // end getCNodeAtLink

/**
 *   method link
 */
public Link link (String name) {
Node tnode = DataSea.find_node_named(name);
return(this.link(tnode, "", Link.DEFAULT_CS, false, "polarized"));
} // end of link()

public Link link (Node node) {
return(this.link(node, "", Link.DEFAULT_CS, false, "polarized"));
} // end of link()

public Link link (Node node, String polarized) {
return(this.link(node, "", Link.DEFAULT_CS, false, polarized));
} // end of link()

//public Link link (Node node, String desc) {
//return(this.link(node, desc, Link.DEFAULT_CS, false, true));
//} // end of link()
```

```
public Link link (Node node, Node CNode) { // link this to node, add CNode to
link
Link link = this.link(node, "", Link.DEFAULT_CS, false, "polarized");
if (CNode != null)
        link.addCNode(CNode);
return(link);
} // end of link()

public Link link (Node node, Node CNode, String polarized) { // link this to
node, add CNode to link
Link link = this.link(node, "", Link.DEFAULT_CS, false, polarized);
if (CNode != null)
        link.addCNode(CNode);
return(link);
} // end of link()

// MAIN ROUTINE FOR LINK, Called by other overloaded versions
public Link link (Node node, String desc, double CS, boolean OneWay, String
polarized) {
int i=0, size;
Link temp_link;
Node tnode;
boolean isPolarized=true;

if (polarized != null) {
        if (polarized.equalsIgnoreCase("polarized"))
                isPolarized = true;
                }

if (node == null) {
        return((Link)null);
        }
if (node.Links.size() >= MAX_CHILDREN ||
        this.Links.size() >= MAX_CHILDREN) {
        return((Link)null);
        }
//
//
// See if already linked
size = this.Links.size();
// this.ChildCount = this.Links.size();
for (i=0; i<size; i++) {
        tnode = this.getNodeAtLink(i);
        if (tnode == node) {
                return((Link)this.Links.elementAt(i));
                }
        }
temp_link = new Link();
temp_link.NodeL = this;
temp_link.NodeR = node;
temp_link.isPolarized = isPolarized;
temp_link.set_CS(CS);
this.set_Desc(node, desc);
this.Links.addElement(temp_link);

if (!OneWay)
        node.Links.addElement(temp_link);

if (DataSea.currentCNode != null) {
        temp_link.addCNode(DataSea.currentCNode);
        //System.out.print("adding DataSea.currentCNode to link for
<"+this.Name+"> and <"+node.Name+">");
        }

return(temp_link);
} // end of link()


/**
 *   method unlink_both    Remove both links between this and given node
 */
        public void unlink_both (Node node) {
                int i=0, size;
        this.unlink(node);
        node.unlink(this);
        return;
} // end of unlink_both()

/**
 *   method unlink    Unlink the argument 'node'.
 *                 Given the node 'node' which is linked  to 'this',
 *                 find the link of 'node' having 'this' as the other half,
 *                 and then remove that link from 'node'. 'this' is unaffected.
 */
        public void unlink (Node node) {
                int i=0, size;
                 Node tnode;
                if (node == null) {
                        return;
                        }
         size = node.Links.size();
                // System.out.print("unlink from this="+this.Name+",
 size="+size+" ... node="+node.Name);
        for (i=0; i<size; i++) {
                tnode = node.getNodeAtLink(i);
                // System.out.print(" "+i+"("+tnode.Name+")");
                if (tnode == this) {// This is the right index
                        // System.out.println(" removing link in "+node.Name+".");
                        node.Links.removeElementAt(i);
                        System.out.println("removed link from "+this.Name+" ->
"+node.Name);
                        return; // Early return
                        }
                }
        return;
} // end of unlink()


/**
 *   method unlink_all    Unlink all nodes from 'this'.
 *                This is probably in preparation for deleting this node.
 */
        public void unlink_all () {
                int i=0, size;
                 Node tnode;

         size = this.Links.size();
        for (i=0; i<size; i++) {
//              if (size == this.Links.size()) // thread problems????
//                      tnode = this.getNodeAtLink(i);
//              else {
//                      GUI.WARNING(0, "Node.unlink_all",
//                              "FAILED, size != old size of Links vector");
//                      return;
//                      }
                tnode = this.getNodeAtLink(i);
                if (tnode == null) {
                        System.out.println("unlink_all: FAILED on tnode, null node
from this.getNodeAtLink("+i+")");
                        }
                else {
                this.unlink_both(tnode);
                }
        }
        return;
} // end of unlink_all()


/**
 *   method undo   returns the old mag
 */
        double undo (int levels) {
        double prior_mag=0;;

        prior_mag = this.mag;


        if (this.magTS >= GUI.lastCommandTS) {
                this.mag = this.mag1;
                this.TS = this.mag1TS;
                this.mag1TS = this.mag2TS;
                this.mag2TS = this.mag3TS;
                }
        if (this.mag1TS >= GUI.lastCommandTS) {
                this.mag = this.mag2;
                this.mag2TS = this.mag3TS;
                }

/*********************************
if (this.Name.equals("Bob") || this.Name.equals("name"))
                System.out.println("undo: "+this.Name+".Tdiff="+(GUI.current_TS-
this.TS)
                +", Mags("+this.mag+","+this.mag1+","+this.mag2+","+this.mag3
  +")");
*********************************/
/*********************************
        if (levels == 1) {
        this.mag = this.mag1;  this.magTS = this.mag1TS;
        this.mag1 = this.mag2;  this.mag1TS = this.mag2TS;
        this.mag2 = this.mag3;  this.mag2TS = this.mag3TS;
        }
        else
        if (levels == 2) {
        this.mag = this.mag2;  this.magTS = this.mag2TS;
        this.mag1 = this.mag3;  this.mag1TS = this.mag3TS;
        }
*********************************/

        this.delta_mag = this.mag - prior_mag;
        if (this.dist > 0)
                this.importance = this.mag/this.dist;
        this.set_TS();

        return (prior_mag);
 } // end of undo()


 /**
  *   method set_TS   returns the old TS
  */
        long set_TS () {
        long old_TS;

        old_TS = this.TS;
        this.TS = GUI.thisCommandTS;
        return(old_TS);
 } // end set_TS
```

```
/**
 *   method TS_diff   returns the the millisecond value of (GUI.thisCommandTS -
this.TS)
 */
        double TS_diff () {

        double diff = GUI.thisCommandTS - this.TS;
//System.out.println("TS_diff("+this.Name+")= "+diff);
        return(diff);
} // end TS_diff

/**
 *   method set_theta_offset
 */
void set_theta_offset (double theta_offset) {
        this.set_theta_offset(theta_offset, "Unknown caller");
} // end set_theta_offset

/*
 *   method set_theta_offset
 */
void set_theta_offset (double theta_offset, String str) {
if (this.Debug && GUI.Debug == 1)
        System.out.println("set_theta_offset(): "+str);
this.ThetaOffset = theta_offset;
} // end set_theta_offset


/**
 **  calc_new_mag
 **
*/
public double calc_new_mag (double current_mag, double given_new_mag) {

double temp_mag = Math.exp(current_mag) + given_new_mag;
if (temp_mag < 2.8)
        temp_mag = 2.8;

double result_mag = Math.log(temp_mag);

System.out.println(current_mag+" + "+given_new_mag+" -> "+result_mag);

return(result_mag);

} // end calc_new_mag


/**
 *   method set_mag_no_history   returns the old mag
 */
double set_mag_no_history (double new_mag) {

// double temp_new_mag = calc_new_mag(this.mag, new_mag);

if (new_mag < this.min_mag) // sanity check
       new_mag = this.min_mag;

if (this.Debug && GUI.Debug == 1)
       System.out.println("set_mag_no_history():"+this.Name+": mag "+this.mag+"
-> "+new_mag);

if (!GUI.doNormalization)  // if we don't normalize, then limit the max value of
mag
       new_mag = (new_mag > MAX_MAG) ? MAX_MAG : new_mag;

this.mag = new_mag;

return(old_mag);
} // end set_mag_no_history


/**
 *   method set_mag   with node argument, use CS to node
 */
double set_mag (double new_mag, Node node) {
Link link = null;
double temp_mag;

// double temp_new_mag = calc_new_mag(this.mag, new_mag);

if (node != null) {
       link = this.getLinkTo(node);
       temp_mag = (link.CS_R + link.CS_L)/2 * new_mag;
       if (this.Debug)
              System.out.println("set_mag of <"+this.Name+"> by node
<"+node.Name+">");
       }
else
       temp_mag = new_mag;

return(this.set_mag(temp_mag));
} // end two argument version

/**
 *   method set_mag   returns the old mag
 */
       double set_mag (double new_mag) {
       double old_mag = this.mag;
       long diff_TS;
```

```
// double temp_new_mag = calc_new_mag(this.mag, new_mag);

if (this.Debug)
        System.out.println("set_mag():"+this.Name+": mag "+this.mag+" ->
"+new_mag);



// Set the elements of the stack of old mags if set_mag
// has not occurred recently
        if ((this.TS + 1000) < GUI.thisCommandTS) {
                this.mag3 = this.mag2; this.mag3TS = this.mag2TS;
                this.mag2 = this.mag1; this.mag2TS = this.mag1TS;
                this.mag1 = this.mag; this.mag1TS = this.magTS;
                                      this.magTS = GUI.thisCommandTS;
                }

this.set_TS();

/************************************************************/
/////////////
// THIS SHOULD BE HANDLED IN HIGHER-LEVEL FUNCTIONS, LIKE more_mag()
// Do Potentiation calculations ...
diff_TS = (GUI.current_TS - this.potentiation_TS);
if (diff_TS < 2000)
        diff_TS = 2000;
if (diff_TS < 4000) { //  -1 <= potentiation <= 1
        // this.potentiation_mag = this.potentiation * 4000/diff_TS;
        this.potentiation_mag = this.potentiation * 2.4; // simpler for now
4/19/2000
        System.out.println("set_mag():            POTENTIATION   "
                +".potentiation_mag = " +this.potentiation_mag
                +", mag=" +this.mag
                +"   "+this.Name);
        }
else
        this.potentiation_mag = 1; // else reset it
/////////////
/************************************************************/

if (this.Debug)
        System.out.println("set_mag():"+this.Name+": mag "+this.mag+" ->
"+new_mag);


if (this.potentiation_mag < 0)
        new_mag /= -this.potentiation_mag; //
else
        new_mag *= this.potentiation_mag; //
```

```
if (new_mag < this.min_mag)
        new_mag = this.min_mag;

if (!GUI.doNormalization)  // if we don't normalize, then limit the max value of
mag
        new_mag = (new_mag > MAX_MAG) ? MAX_MAG : new_mag;

this.mag = new_mag; // necessarily update this.mag

this.delta_mag = new_mag - this.mag1; // delta dependent on mag1 which is dep'
on TS

return (old_mag);
} // end of set_mag()




double lift_to_threshold () {
double new_mag=this.mag;

if (new_mag < GUI.relations_threshold+0.1)
        this.set_mag(GUI.relations_threshold+0.1);

return(new_mag);
} // end lift_to_threshold


double lift (double delta) {
double new_mag=this.mag;

new_mag += delta;
if (new_mag >= MAX_MAG)
        new_mag = MAX_MAG;
this.set_mag(new_mag);
return(new_mag);
} // end lift


double a_little_more_mag () {
double new_mag=1;
new_mag = 1.3*this.mag;
this.set_mag(new_mag);
return(new_mag);
} // end a_little_more_mag
```

```
double a_little_less_mag () {
double new_mag=1;
new_mag = this.mag/1.1;
this.set_mag(new_mag);
return(new_mag);
} // end a_little_less_mag


/**
 *   method more_mag   returns the old mag
 */
        double more_mag () {
return(this.more_mag((Node)null));
}

        double more_mag (Node node) { // the link is from 'node' to 'this'
double new_mag=MIN_MAG;
Link link = null;

if (node != null) {
        link = this.getLinkTo(node);
        new_mag = (link.CS_R + link.CS_L)/2 + this.mag;
        }
else
        new_mag = 1 + this.mag;

// Set a minimum value for this node's mag
new_mag = (new_mag < BIG_MAG) ? BIG_MAG : new_mag;

if (this.Debug)
        {
        if (node==null)
        System.out.println("more_mag(): from null node to this='"+this.Name+"',
new_mag="+new_mag);
        else
        System.out.println("more_mag(): from node='"+node.Name+"' to
this='"+this.Name+"', new_mag="+new_mag);
        }

return (set_mag(new_mag, node));
} // end of more_mag()


/**
 *   method less_mag   returns the old mag
 */
        double less_mag () {
```

```
return(this.less_mag((Node)null));
}
        double less_mag (Node node) { // the link is from 'node' to 'this'
double new_mag=MIN_MAG;

new_mag = this.mag/1.2;
if (new_mag < MIN_MAG)
        new_mag = MIN_MAG;

return (set_mag(new_mag));
} // end of less_mag()



/**
 *   method more_CS   returns the new CS
 */
public void more_CS (Node node) {

Link link = this.getLinkTo(node);
if (this.Debug && GUI.Debug == 1)
        System.out.println("more_CS() between this=<"+this.Name+"> and
node=<"+node.Name+">");
link.more_CS();

return;
} // end of more_CS()



/**
 *   method less_CS   returns the new CS
 */
public void less_CS (Node node) {

Link link = this.getLinkTo(node);
link.less_CS();

return;
} // end of less_CS()


/**
 *   method set_dist   returns the old dist
 */
double set_dist (int new_dist) {
double tdist, new_double_dist;

new_double_dist = (double)new_dist;
```

```
return(this.set_dist((double)new_double_dist));
}

double set_dist(double new_dist) {
double tdist;

/**************************************************************************
*************/
// Thanks to java's imprecision, I need to manipulate double to eliminate
0.00000000002 error
/**************************************************************************
*************/
if (new_dist > 0) {
tdist = ((int)(10.0*(new_dist + 0.000001)))/10.0;
if (tdist != new_dist)
       new_dist = tdist;
}
/**************************************************************************
*************/

this.old_dist = this.dist;
this.dist = new_dist;

if (new_dist > 0)
       this.importance = this.mag/this.dist;
//this.Tdist = this.dist; // Set this as default

return (this.old_dist);

} // end of set_dist()




/**
 *   method set_Tdist
 */
void set_Tdist (int new_Tdist) {
this.Tdist = new_Tdist;
return;
}


/**
 *   method setData
 */
       int setData (String data) {
    this.Data[0] = data;
       return (0);
} // end of setData()
```

```
/**
 *   method action
 */
        public void action () {
    if (this.Desc.equals("File_mgt"))
        this.file_mgt();
        return;
} // end of action()

 /**
 **  Node.file_mgt
 **
*/
        public void file_mgt () {
    GUI.P(0,"Node.file_mgt","Operating on "+this.Name);
} // end file_mgt


 /**
 *   method describe
 */
        public String describe (String ToWhom) {
        if (ToWhom == "To Console") {
                return((String) null);
                }
        if (ToWhom == "To Me") {               }
          // Default
    return("Name=" +this.Name+ ": Type=" +this.Type+ ", Desc=" +Desc +"'");

} // end of describe()

/**
 **
 */
public double msg (String action, String msg, long TS, long dt) {
long deadline_millis;

deadline_millis = java.lang.System.currentTimeMillis() - (TS + dt);
// if (deadline_millis > 0)
//      GUI.P(0, "Node.msg", "clock expired, " +deadline_millis+ " milliseconds
over.");
// else
//      GUI.P(0, "Node.msg", "clock not expired, "
//   +deadline_millis+ " milliseconds over.");

        return(0.0);
}
```

```
/*
**  Try handling various recursions, do something along the network
**/
float network_start (Node node){

if (node == null)
   return(-1);


node.dist = 0;

network_recursive(node);
return(0);
} // end position_start (a simple function calling recursive fns)



/*
**  Try handling various recursions, do something along the network
**/
float network_recursive (Node node){

if (node == null)
   return(-1);

node.dist = 0;

return(0);
} // end position_start (a simple function calling recursive fns)


/*************************************************
public class DataObj extends Object {
       // some elements for data, stubbed now
       public String getDataAsString () { return((String)null);};
} // end class DataObj
*************************************************/


}       // End of class Node
```

```
// This is Force.java     by Rocky Nevin

import java.lang.Math;

public class Force extends Object {
double pxPOV, pyPOV;
double thetaOffset = 0.1;
double pxParent, pyParent;
double pxSiblings, pySiblings;
double pxNeighbors, pyNeighbors;
double pxWind, pyWind;
double pxSum, pySum;
double ThetaRange=1, theta=1.0, x=0, y=0, dist=0;
int i;
double noiseCounter=0, xNoise=0, yNoise=0;
long local_TS;
boolean XNeg=false, YNeg=false;
double opt_dist, alpha, beta;
static double rand_scale = 10/GUI.magscale; // guages severity of heat noise

static boolean calcPOVPressure=false;
static boolean calcNeighborPressure=false;
static boolean calcParentPressure=true;
static boolean calcNoise=false;


/**
 **  shift_proximally  shift proximal nodes by X,Y
 **  This is expensive, called by calc_pressures each time shifted AN/DN is
found
*/
public void shift_proximally (Node node, double X, double Y) {
int i, size;
Node child;

if (node.dist <= 3) // don't go back too far
        return;

size = node.Links.size();
for (i=0; i<size; i++) {
        child = (Node)(node.getNodeAtLink(i));
        if (child.dist < node.dist) {
                child.x = X;
                child.y = Y;
                shift_proximally(child, X, Y);
                }
}


} // end shift_proximally
```

```
  /**
   ** calc_pressures
   **  Put calculated values into force object, apply these to child node
   **
   */
   public double calc_pressures (Node parent, Node child) {
   double dx, dy, theta, dist;
   double delta_x, delta_y, distsq;
   double desired_x, desired_y, diff, desired_dist=.2, desired_theta=0;
   double temp_dist;
   int i, size;
   int ChildCount=0, ChildNum=0;
   int grand_parent_ChildCount=4;
   Node tnode=null, grand_parent_node;

if ((parent==null) || (child==null)) {
     GUI.WARNING(0,"calc_pressures", "NULL: parent="+parent+", child="+child);
     return(-1);
     }
   pxParent=0;pyParent=0;
   pxPOV=pyPOV=0;
   pxNeighbors=pyNeighbors=0;
   pxWind=pyWind=0;
   pxSum=pySum=0;

if (child.isFrozen)
     return(0);

/*******************************************************************************
******
// This is expensive, called by calc_pressures each time shifted AN/DN is found
   if (!child.isLayer)
     if (GUI.VRyes(child))  {
          child.x = child.X;
          child.y = child.Y;
          shift_proximally(child, child.X, child.Y);
          }
********************************************************************************
**********/


   if (child.mag < GUI.pos_threshold)
    return(0);
   if (child.mag <= 0.2)
    return(0);

/****************************************************
** // BEGIN WIND PRESSURE BACK TO DATASEA
**      if (child.y<0)
**         pyWind=.1;
*************************************************/
if (GUI.drawWind && child.BigChildCount>2)
        pyWind = child.BigChildCount * 3.0;

if (GUI.mode_obj.position_mode.equals("Grid"))
        {
        temp_dist = (double)(child.dist);
        if (temp_dist <= 0.0)
                temp_dist = 1.0;
        desired_x = 150-50*(temp_dist); // compresses towards the left
//      desired_y = -100 + 10*child.mag;
        pxPOV = (desired_x - child.x);
//      pyPOV = (desired_y - child.y);
        }
else
if (GUI.mode_obj.position_mode.equals("Lvls"))
        {
/*********************************************************
*        delta_y = GUI.datasea.POV.y - child.y;
*        dist = Math.sqrt(delta_y * delta_y);
*       alpha = 50;
*       opt_dist = alpha * child.dist;
*       desired_x = parent.x+10*alpha*(child.ChildNum-1-
0.5*parent.ChildCount)/(double)(parent.ChildCount * child.dist);
*       desired_y = GUI.datasea.POV.y + opt_dist;
*       pxPOV = desired_x - child.x;
*       pyPOV = desired_y - child.y;
**********************************************************/

// NEW THING ... Organize nodes by Node.dist, URLs by Node.mag separated into
two half-fields
if (!child.isAN) { // is not an AN
        desired_x =  20+10*(double)(Node.MAX_MAG - child.mag); // compresses
towards the right
        } else { // is AN
                //temp_dist = (double)(child.Tdist); // - 1);
                temp_dist = (double)(child.AN_level);
                if (temp_dist <= 0.0)
                        temp_dist = 1.0;
                desired_x = -50*(temp_dist); // compresses towards the left
                }
//      desired_y = parent.y + 10*child.ChildNum;
        pxPOV = (desired_x - child.x);
//      pyPOV = (desired_y - child.y);
        } // END NEW LEVEL POSITIONING

else if (GUI.mode_obj.position_mode.equals("Rel'"))
{ // START RELATIVE POSITIONING
```

```
// BEGIN POV PRESSURE  // Optimal dist to POV is 50+300/child.mag
 if (child.mag > 1) { // DON"T BOTHER IF TOO SMALL, THIS IS AN EXPENSIVE
OPERATION
   if ((calcPOVPressure==true) &&(GUI.datasea.POV != null)) {
        delta_x = GUI.datasea.POV.x - child.x;
        delta_y = GUI.datasea.POV.y - child.y;
        theta = GUI.get_angle(delta_x, delta_y);
        distsq = (delta_x*delta_x)+(delta_y*delta_y);
        dist = Math.sqrt(distsq);

        pxPOV = (dist-50-300./child.mag) * .1* Math.cos(theta);
        pyPOV = (delta_y+50+300./child.mag); // * Math.sin(theta);
     }
} // end POV PRESSURE

// BEGIN PARENT PRESSURE
   if (calcParentPressure==true) {
        {
        grand_parent_node = parent.getParent();
        if (grand_parent_node != null)
               grand_parent_ChildCount = grand_parent_node.ChildCount;
        else
               grand_parent_ChildCount = 4;

        if (parent == GUI.datasea.POV)
               desired_dist = 10;
        else
               desired_dist = parent.mag * 40/parent.dist; // 7/26/99
        if (child.isAN)
               desired_dist += desired_dist; // double dist for ANs
        ChildCount = parent.ChildCount;  // 6/8/99
        ChildNum = child.ChildNum;  // 6/8/99
        ThetaRange = GUI.spread_factor/(double)(grand_parent_ChildCount);
        desired_theta = ThetaRange*ChildNum + parent.ThetaOffset;
        desired_theta -= 0.2;
        desired_theta = GUI.spread_factor*(ChildNum - ChildCount/2) +
parent.ThetaOffset;
        //desired_theta += (3.0/child.mag)*thetaOffset; // make spirals   6/21/99


        if (parent.isURL) { // 8/24/99
               desired_dist *= child.mag/4.0;
               }
        else
        if (parent.isBB) { // 8/24/99
               desired_dist *= child.mag/4.0;
               }
        else
        if (parent.isDN) {
```

```
                    desired_dist *= child.mag/25.0;
                    }
               child.ThetaOffset=desired_theta;
               //desired_x = parent.x + desired_dist * Math.sin(desired_theta);  //
      6/8/99
               //desired_y = parent.y + desired_dist * Math.cos(desired_theta);  //
      6/8/99
               desired_x = parent.x + desired_dist * Math.cos(desired_theta);  // 6/8/99
               desired_y = parent.y + desired_dist * Math.sin(desired_theta);  // 6/8/99
               pxParent = 0.5*(desired_x - child.x);  // 6/8/99
               pyParent = 0.5*(desired_y - child.y);  // 6/8/99
               }


      /************************************
      if (child.isDN) {
               }
      ************************************/

         }
      } // END RELATIVE POSITIONING

      // BEGIN NEIGHBOR PRESSURE
               if (calcNeighborPressure==true) {
               // DON"T BOTHER IF TOO SMALL, THIS IS AN EXPENSIVE OPERATION
       if ((child.mag >= GUI.pos_threshold) && (!child.isLayer)) {
               {
               size = GUI.datasea.node_vec.size();
               for (i=0; i<size; i++) {
                   tnode = (Node)GUI.datasea.node_vec.elementAt(i);
                   if ((tnode != child)
                      && ((tnode.isAN && child.isAN)
                      || (!tnode.isAN && !child.isAN))) {
      //System.out.print(tnode.Name+" ");
                   delta_x = tnode.x - child.x;
                   delta_y = tnode.y - child.y;
                   distsq = ( (delta_x*delta_x) + (delta_y*delta_y));
                   desired_dist = 3*(tnode.size_y + child.size_y);
                   if (distsq < (desired_dist*desired_dist)) {
                       theta = GUI.get_angle(delta_x, delta_y);
                       dist = Math.sqrt(distsq);
                      if (!GUI.mode_obj.position_mode.equals("Lvls"))
                           pxNeighbors -= (desired_dist-dist)*Math.cos(theta);
                       pyNeighbors -= 3*(desired_dist-dist)*Math.sin(theta);
                       }
                       }
                   }
               }
               }
         }
      // maybe_noise();

      pxSum = pxParent + pxPOV + pxNeighbors + pxWind + xNoise;
      pySum = pyParent + pyPOV + pyNeighbors + pyWind + yNoise;
      if (pxSum < 0)
           pxSum = -Math.sqrt(-pxSum);
      else
           pxSum = Math.sqrt(pxSum);
      if (pySum < 0)
           pySum = -Math.sqrt(-pySum);
      else
           pySum = Math.sqrt(pySum);

           if (pxSum > GUI.globalMaxPressure)
                  GUI.globalMaxPressure = pxSum;
           if (pySum > GUI.globalMaxPressure)
                  GUI.globalMaxPressure = pySum;

      return(0);
      } // End of Force.calc_pressures


      /**
      ** noise    added to image to avoid stable points and show aliveness
      */
   public void maybe_noise () {
   if (!calcNoise) {
       xNoise=0;
       yNoise=0;
       return;
       }

      if (noiseCounter==0) {
       xNoise = rand_scale*(GUI.random()-.5);
       yNoise = rand_scale*(GUI.random()-.5);
       }
      else {
       if (noiseCounter>11)
           noiseCounter=-1;
           xNoise=0;
           yNoise=0;
           }
       noiseCounter ++;


       xNoise = rand_scale*(GUI.random()-.5);
       yNoise = rand_scale*(GUI.random()-.5);
   } // end noise
   } // end of Object Force
```

```
// This is GetURLInfor.java     by Rocky Nevin
// Started on Flanagan, heavily modified

// This example is from the book _Java in a Nutshell_ by David Flanagan.
// Written by David Flanagan.  Copyright (c) 1996 O'Reilly & Associates.
// You may study, use, modify, and distribute this example for any purpose.
// This example is provided WITHOUT WARRANTY either expressed or implied.

import java.net.*;
import java.io.*;
import java.util.*;

// Return an array of Stings holding the data
// Parse the path minus the file name, use that to follow hyperlinks
// Create nodes for each URL, link them


public class GetURLInfo {
static String str_array[];
static int MAX_LINES = Node.MAX_TEXT_DATA_LINES;

    public static void printinfo(URLConnection u) throws IOException {
        int index=0, end=0;
        // Display the URL address, and information about it.
        if (u == null) {
                System.out.println("GetURLInfo.printinfo(): connection is null");
                return;
                }
        else {;}
/************************************************************************
        System.out.println("GetURLInfo.printinfo(): connection is <"+u+">");
        System.out.println(u.getURL().toExternalForm() + ":");
        System.out.println("  Content Type: " + u.getContentType());
        System.out.println("  Content Length: " + u.getContentLength());
        System.out.println("  Last Modified: " + new Date(u.getLastModified()));
        System.out.println("  Expiration: " + u.getExpiration());
        System.out.println("  Content Encoding: " + u.getContentEncoding());
        System.out.println("  getHeaderField(0): "+u.getHeaderField(0));
        System.out.println("  getHeaderField(1): "+u.getHeaderField(1));
        System.out.println("  getHeaderField(2): "+u.getHeaderField(2));
        System.out.println("  getHeaderField(3): "+u.getHeaderField(3));
        System.out.println("  getHeaderField(4): "+u.getHeaderField(4));
        System.out.println("  getHeaderField(5): "+u.getHeaderField(5));
        System.out.println("  getHeaderField(6): "+u.getHeaderField(6));
        System.out.println("  getHeaderField(7): "+u.getHeaderField(7));
        System.out.println("  getHeaderField(8): "+u.getHeaderField(8));

************************************************************************/
        // Read and print out the first MAX_LINES lines of the URL.
        System.out.println("First "+MAX_LINES+" lines:");
        DataInputStream in = new DataInputStream(u.getInputStream());
        for(int i = 0; i < MAX_LINES; i++) {
            String line = in.readLine();
               str_array[i] = line;
            if (line == null) break;
/**********************************************************************
               index = line.toLowerCase().indexOf("href");
               if (index > 0) {
                       index += 6;
                       end = line.toLowerCase().indexOf("\"", index+1);
                       if (end>index && end < line.length())
                       System.out.println(" =====> URL? <" +
line.substring(index, end)+">");
                       else
                       System.out.println(" =====> PROBLEM: index="+index+",
end="+end);
                       }
            System.out.println("i="+i+") " + line);
**********************************************************************/
        }
    }

    // Create a URL from the specified address, open a connection to it,
    // and then display information about the URL.
    public String[] get_URL(String name, int max_lines)
        throws MalformedURLException, IOException
    {
        MAX_LINES = max_lines; // OVERRIDE THE DEFAULT HERE
        System.out.println("get_URL: MAX_LINES is now "+MAX_LINES);
        return(get_URL(name));
        } // end of max_lines version of get_URL()

    public String[] get_URL(String name)
        throws MalformedURLException, IOException
    {
        str_array = new String[MAX_LINES];
        URL url = new URL(name);
        URLConnection connection = url.openConnection();

System.out.println("GetURLInfo.getURL(): getHost(): "+url.getHost());
System.out.println("GetURLInfo.getURL(): getRef(): "+url.getRef());
System.out.println("GetURLInfo.getURL(): toString(): "+url.toString());

if ((0<name.toLowerCase().indexOf("http")) && !GUI.NetOK) {
        System.out.println("GetURLInfo.getURL(): GUI.NetOK is false, aborting
data retrieval.");
        return(str_array);
        }
        printinfo(connection);
        return(str_array);
```

```
      }
  }
```

```
// This is Input.java

import java.io.*;
import java.util.*;
import java.lang.*;

public class Input extends Object {
GUI gui; // The GUI passed to the Input constructor


public Input (GUI passed_gui_object) {        // Constructor
gui = passed_gui_object;
}

/*
 *
 *
 */
   // This method breaks a specified label up into an array of words.
   // It uses the StringTokenizer utility class.
   // From Nutshell, chapter 5, Multi*.java
   public Node string_input (String input_string) {
      int num_words;
      String words[];
      String tempStr;
      Node new_node, ret_node=null;
      double event_offset = 0;
      String event_string="";

       GUI.P(2,"datasea.string_input", "<"+input_string+">");
      if (input_string.equals(".")) {
             tempStr = GUI.priorCommand;
             GUI.priorCommand = "";
             return(ret_node=string_input(tempStr)); // re-invoke
             }
      GUI.priorCommand = input_string; // for use above, later

 // StreamTokenizer can handle converting strings to numbers, but can't count
 tokens
 // StringTokenizer can not convert but can count.
 // The number result from StreamTokenizer can't easily be converted back into a
 string.
 // Its all unbelievably stupid and complex.

 // StringReader r = new StringReader(input_string);
 // StreamTokenizer t = new StreamTokenizer(r);

        StringTokenizer st = new StringTokenizer(input_string, " ,.<>\"\t\r\n"
 );
        num_words = st.countTokens();
```

```
            words = new String[num_words];

            event_offset = 0;
            // accumulated_event_offset = 0;
            for(int i = 0; i < num_words; i++) {
                    words[i] = st.nextToken();
                    }

            GUI.lastCommandTS = GUI.thisCommandTS;
            GUI.thisCommandTS = GUI.current_TS;
            ret_node = word_selector(words, input_string, num_words);

            gui.datasea.normalize();

    return(ret_node);
    } // end string_input

        public Node word_selector (String[] words, String input_string, int
    num_words) {
            Node new_node, ret_node=null;
            String command;
            boolean understood = true;

            if (num_words >= 1)
                    command = words[0];
            else {
                    return(ret_node);
            }

            GUI.P(1,"word_selector","num_words="+num_words+", input="+input_string);

             if (num_words == 0)
                 return(ret_node);
            else
             if (command == null)
                 return(ret_node);
             GUI.P(1,"datasea.word_selector", input_string);

    // COMMANDS Commands commands
             if (command.equals("net") ) { //
                    gui.NetOK = !gui.NetOK;
                    gui.P(0,"word_selector","NetOK = "+gui.NetOK);
                 }
             else
             if (command.equals("t0") ) { //
                    gui.max_transition_count = 0;
                    GUI.P(0,"datasea.word_selector",
    "gui.max_transition_count="+gui.max_transition_count);
                 }
             else
        if (command.equals("t1") ) { //
                gui.max_transition_count = 1;
                GUI.P(0,"datasea.word_selector",
"gui.max_transition_count="+gui.max_transition_count);
           }
        else
        if (command.equals("t2") ) { //
                gui.max_transition_count = 2;
                GUI.P(0,"datasea.word_selector",
"gui.max_transition_count="+gui.max_transition_count);
           }
        else
        if (command.equals("t3") ) { //
                gui.max_transition_count = 3;
                GUI.P(0,"datasea.word_selector",
"gui.max_transition_count="+gui.max_transition_count);
           }
        else
        if (command.equals("t4") ) { //
                gui.max_transition_count = 4;
                GUI.P(0,"datasea.word_selector",
"gui.max_transition_count="+gui.max_transition_count);
           }
        else
        if (command.equals("t5") ) { //
                gui.max_transition_count = 5;
                GUI.P(0,"datasea.word_selector",
"gui.max_transition_count="+gui.max_transition_count);
           }
        else
        if (command.equals("t6") ) { //
                gui.max_transition_count = 6;
                GUI.P(0,"datasea.word_selector",
"gui.max_transition_count="+gui.max_transition_count);
           }
        else
        if (command.equals("t7") ) { //
                gui.max_transition_count = 7;
                GUI.P(0,"datasea.word_selector",
"gui.max_transition_count="+gui.max_transition_count);
           }
        else
        if (command.equals("g1") ) { //
           gui.datasea.group(gui.datasea.POV,1);
           }
        else
        if (command.equals("g2") ) { //
           gui.datasea.group(gui.datasea.POV,2);
           }
        else
```

```
    if (command.equals("g3") ) { //
      gui.datasea.group(gui.datasea.POV,3);
      }
    else
    if (command.equals("g4") ) { //
      gui.datasea.group(gui.datasea.POV,4);
      }
    else
    if (command.equals("max") ) { // find maximum mag node
      gui.datasea.find_max();
      }
    else
    if (command.equals("rel") ) { // release a node (its link) from the POV
      gui.datasea.word_release(words, num_words);
      }
    else
    if (command.equals("freeze") ) { //
      gui.datasea.freeze();
      }
    else
    if (command.equals("unfreeze") ) { //
      gui.datasea.unfreeze();
      }
    else
    if (command.equals("flip") ) { //
      gui.FlipAxes = !gui.FlipAxes;
      }
    else
    if (command.equals("bound") ) { //
          gui.showBoundaries = !gui.showBoundaries;
          GUI.P(0,"datasea.word_selector","showBoundaries is
"+gui.showBoundaries);
          if (gui.showBoundaries)
                 gui.datasea.set_Tdist_start(gui.lastNode);
      }
    else
    if (command.equals("norm") ) { //  shall we normalize?
      gui.doNormalization = !gui.doNormalization;
      gui.status("doNormalization is "+gui.doNormalization);
      }
    else
    if (command.equals("wind") ) { //  shall we use wind?
      gui.drawWind = !gui.drawWind;
      gui.status("drawWind is "+gui.drawWind);
      }
    else
    if (command.equals("da") ) { //  shall we draw Abstract Nodes?
      gui.drawAN = !gui.drawAN;
      gui.status("drawAN is "+gui.drawAN);
      }
```

```
else
 if (command.equals("don") ) { // shall we draw Object Nodes?
    gui.drawON = !gui.drawON;
   gui.status("drawON is "+gui.drawON);
    }
else
 if (command.equals("dp") ) { // shall we draw P Nodes?
    gui.drawPN = !gui.drawPN;
   gui.status("drawPN is "+gui.drawPN);
    }
else
 if (command.equals("dc") ) { // shall we draw Context Nodes?
    gui.drawCN = !gui.drawCN;
   gui.status("drawCN is "+gui.drawCN);
    }
else
 if (command.equals("dl") ) { // shall we draw Link Names?
    gui.drawLinkNames = !gui.drawLinkNames;
   gui.status("drawLinkNames is "+gui.drawLinkNames);
    }
else
 if (command.equals("de") ) { // shall we draw Event Nodes?
    gui.drawEvent = !gui.drawEvent;
   gui.status("drawEvent is "+gui.drawEvent);
    }
else
 if (command.equals("du") ) { // shall we draw Data Nodes?
    gui.drawURL = !gui.drawURL;
   gui.status("drawURL is "+gui.drawURL);
    }
else
 if (command.equals("dd") ) { // shall we draw Data Nodes?
    gui.drawDN = !gui.drawDN;
   gui.status("drawDN is "+gui.drawDN);
    }
else
 if (command.equals("df") ) { // shall we draw Data Nodes?
    gui.drawFile = !gui.drawFile;
   gui.status("drawFile is "+gui.drawFile);
    }
else
 if (command.equals("con") ) { //
   gui.datasea.con();//mag from the context node connected to lastNode
    }
else
 if (command.equals("SS") ) { //
   gui.datasea.SS(words, num_words);
    }
else
 if (command.equals("mail") ) { //
```

```
        gui.datasea.word_mail(words, num_words);
         }
    else
     if (command.equals("find") ) { //
        gui.datasea.word_select(words, num_words);
         }
    else
     if (command.equals("AN") ) { //
        gui.datasea.word_select(words, num_words);
         }
    else
     if (command.equals("DN") ) { //
        gui.datasea.word_select(words, num_words);
         }
    else
     if (command.equals("select") ) { //
        gui.datasea.word_select(words, num_words);
         }
    else
     if (command.equals("un") ) { //
        gui.datasea.word_unselect(words, num_words);
         }
    else
     if (command.equals("unselect") ) { //
        gui.datasea.word_unselect(words, num_words);
         }
    else
     if (command.equals("focus") ) { //
        gui.datasea.word_focus(words, num_words);
         }
    else
     if (command.equals("rename") ) { //
        gui.datasea.word_rename(words, num_words);
         }
    else
     if (command.equals("most") ) { //
        gui.datasea.word_most(words, num_words);
         }
    else
     if (command.equals("ap") ) { // amplify plus
        gui.amplify_region(1);
         }
    else
     if (command.equals("am") ) { // amplify minus
        gui.amplify_region(-1);
         }
    else
     if (command.equals("pl") ) { // position levels
        word_mode_command(words, num_words);
         }
```

```
          else
           if (command.equals("pr") ) { // position relations
              word_mode_command(words, num_words);
               }
          else
           if (command.equals("snap") ) { // render VR (now 'snap')
              word_mode_command(words, num_words);
               }
          else
           if (command.equals("nosnap") ) { //
              word_mode_command(words, num_words);
               }
          else
           if (command.equals("VR") ) { //
                  if (gui.lastNode != null) {
                    gui.lastNode.VRyes = !gui.lastNode.VRyes;
                    GUI.P(0,"word_selector","VRyes of "+gui.lastNode.Name+" is
"+gui.lastNode.VRyes);
                    }
                  else
                    GUI.WARNING(0,"word_selector","Need lastNode for VRyes
setting");
          }
/*************
          else
//           if (command.equals("lr") ) { // lines radial
//            word_mode_command(words, num_words);
//               }
          else
//           if (command.equals("ld") ) { // lines distal
//            word_mode_command(words, num_words);
//               }
          else
//           if (command.equals("lp") ) { // lines proximal
//            word_mode_command(words, num_words);
//               }
*********/
          else
           if (command.equals("sr") ) { // spread radial
              word_mode_command(words, num_words);
               }
          else
           if (command.equals("sd") ) { // spread distal
              word_mode_command(words, num_words);
               }
          else
           if (command.equals("sp") ) { // spread proximal
              word_mode_command(words, num_words);
               }
          else
          if (command.equals("upurls") ) {
            gui.datasea.upurls(words, num_words);
             }
          else
          if (command.equals("shiftOut") ) {
            gui.datasea.shiftOut(words, num_words);
             }
          else
          if (command.equals("shiftIn") ) {
            gui.datasea.shiftIn(words, num_words);
             }
          else
          if (command.equals("shiftInAll") ) {
            gui.datasea.shiftInAll(words, num_words);
             }
          else
          if (command.equals("shiftOutAll") ) {
            gui.datasea.shiftOutAll(words, num_words);
             }
          else
          if (command.equals("shiftOut") ) {
            gui.datasea.shiftOut(words, num_words);
             }
          else
          if (command.equals("heavy1") ) {
            gui.datasea.heavyans((Node)null, gui.lastNode, 1, 0);
             }
          else
          if (command.equals("heavy2") ) {
            gui.datasea.heavyans((Node)null, gui.lastNode, 2, 0);
             }
          else if (command.equals("heavy3") ) { gui.datasea.heavyans((Node)null,
gui.lastNode, 3, 0); } else if (command.equals("heavy4") ) {
gui.datasea.heavyans((Node)null, gui.lastNode, 4, 0); } else if
(command.equals("an1") ) { gui.datasea.mag_ans((Node)null, gui.lastNode, 1, 0);
} else if (command.equals("an2") ) { gui.datasea.mag_ans((Node)null,
gui.lastNode, 2, 0); } else if (command.equals("an3") ) {
gui.datasea.mag_ans((Node)null, gui.lastNode, 3, 0); } else if
(command.equals("an4") ) {
            gui.datasea.mag_ans((Node)null, gui.lastNode, 4, 0);
             }
          else
          if (command.equals("ans") ) {
            gui.datasea.ans();
             }
          else
          if (command.equals("storebig") ) {
            gui.datasea.storebig();
             }
          else
```

```
if (command.equals("x1") ) {
  gui.datasea.storebig();
   }
else
 if (command.equals("x2") ) {
  gui.datasea.flattenANs();
   }
else
 if (command.equals("x3") ) {
  gui.datasea.absURLs();
   }
else
 if (command.equals("x4") ) {
  gui.datasea.inhstored();
   }
else
 if (command.equals("connect") ) {
  gui.datasea.connect_all_to_POV();
   }
else
 if (command.equals("absURLs") ) {
  gui.datasea.absURLs();
   }
else
 if (command.equals("inhstored") ) {
  gui.datasea.inhstored();
   }
else
 if (command.equals("xabs") ) {
  gui.datasea.storebig();
  gui.datasea.flattenANs();
  gui.datasea.absURLs();
  gui.datasea.inhstored();
   }
else
 if (command.equals("selectrecent") ) {
  gui.datasea.selectrecent();
   }
else
 if (command.equals("flattenANs") ) {
  gui.datasea.flattenANs();
   }
else
 if (command.equals("abs") ) {
  gui.datasea.abs(words, num_words);
   }
else
 if (command.equals("abs1") ) {
  gui.datasea.abs(words, num_words);
   }
```

```
        else
        if (command.equals("abs2") ) {
           gui.datasea.abs(words, num_words);
           }
        else
        if (command.equals("abs3") ) {
           gui.datasea.abs(words, num_words);
           }
        else
        if (command.equals("sim") ) {
           gui.datasea.sim(words, num_words, '+');
           }
        else
        if (command.equals("unsim") ) {
           gui.datasea.sim(words, num_words, '-');
           }
        else
        if (command.equals("pump") ) { // pump up all distal nodes
           gui.datasea.pump(gui.lastNode);
           }
        else
        if (command.equals("local") ) {
           gui.datasea.local(gui.lastNode);
           }
        else
        if (command.equals("chain") ) { // mag similar nodes distally
           gui.datasea.chain(gui.lastNode);
           }
        else
        if (command.equals("sides") ) { // strip off AN's, see sides of chains
of data
           gui.datasea.sides(gui.lastNode);
           }
        else
        if (command.equals("org") ) { // Organize children's theta position by
mag
                gui.datasea.theta_org = !gui.datasea.theta_org;
                gui.status("theta_org = "+gui.datasea.theta_org);
                GUI.P(0,"word_selector","theta_org = "+gui.datasea.theta_org);
                }
        else
        if (command.equals("strip") ) { // strip off DN's, see only AN's
           gui.datasea.strip(words, num_words);
           }
        else
        if (command.equals("and") ) {
           gui.datasea.and(words, num_words);
           }
        else
        if (command.equals("backup") ) {
            gui.datasea.backup();
            }
         else
          if (command.equals("pot") ) {
            gui.datasea.pot(words, num_words);
            }
         else
          if (command.equals("potmag") ) {
               Node.potentiation = 1;
            gui.datasea.potmag(words, num_words);
            }
         else
          if (command.equals("npotmag") ) {
               Node.potentiation = -1;
            gui.datasea.potmag(words, num_words);
            }
         else
          if (command.equals("cats") ) {
            gui.datasea.cats(words, num_words);
            }
         else
          if (command.equals("backs") ) {
            gui.datasea.backs(words, num_words);
            }
/****************************************
         else
          if (command.equals("upstream") ) {
            gui.datasea.upstream(words, num_words);
            }
*************************************/
         else
          if (command.equals("back3") ) {
            gui.datasea.back3(words, num_words);
            }
         else
          if (command.equals("backt") ) {
            gui.datasea.backt(words, num_words);
            }
         else
          if (command.equals("back") ) {
            gui.datasea.back(words, num_words);
            }
         else
          if (command.equals("Back") ) {
            gui.datasea.Back(words, num_words);
            }
         else
          if (command.equals("dumpCNs") ) {
            gui.datasea.dumpCNs();
            }
```

```
else
 if (command.equals("showCNs") ) {
    gui.datasea.showCNs();
    }
else
 if (command.equals("showevents") ) {
    gui.datasea.showevents();
    }
else
 if (command.equals("showtime") ) {
    gui.datasea.showtime();
    }
else
 if (command.equals("link") ) { //
   gui.datasea.word_link(words, num_words);
        gui.datasea.needdistUpdate = true;
    }
else
 if (command.equals("unlink") ) { //
   gui.datasea.word_unlink(words, num_words);
        gui.datasea.needdistUpdate = true;
    }
else
 if (command.equals("delete") ) { //
   gui.datasea.word_delete(words, num_words);
        gui.datasea.needdistUpdate = true;
    }
else
 if (command.equals("show") ) {
    gui.datasea.show(input_string);
    }
else
 if (command.equals("setPOV") ) {
        gui.datasea.set_POV();
        gui.datasea.needdistUpdate = true;
    }
else
if (command.equals("one") ) {
    gui.datasea.reset_mags();
     }
else
if (command.equals("distupdate") ) {
    gui.datasea.needdistUpdate = true;
     }
else
if (command.equals("quick") ) {
         gui.quick = !gui.quick;
         gui.P(0,"word_selector","quick = "+gui.quick);
         gui.status("quick = "+gui.quick);
     }
```

```
else
if (command.equals("resetprint") ) {
        GUI.global_str_size = 0;
     }
else
if (command.equals("resetCS") ) {
    gui.datasea.reset_CSs();
     }
else
if (command.equals("save") ) {
    gui.datasea.word_save();
     }
else
if (command.equals("reset") ) {
    gui.datasea.word_reset();
        gui.datasea.needdistUpdate = true;
     }
else
if (command.equals("X") ) {
    gui.set_Xnode();
     }
else
if (command.equals("r") ) {
    gui.datasea.word_reset();
        gui.auto_rescale = true;
        gui.status("auto_rescale = "+gui.auto_rescale);
     }
else
if (command.equals("buttons") ) {
    gui.show_buttons();
     }
else
if (command.equals("lines") ) {
        gui.mode_obj.toggle_lines_mode();
     }
else
 if (command.equals("enh") ) {
    gui.datasea.enhance(words, num_words);
    }
else
 if (command.equals("m1") ) {
    gui.datasea.mag(words, num_words, "both", 1, "+");
    }
else
 if (command.equals("m2") ) {
    gui.datasea.mag(words, num_words, "both", 2, "+");
    }
else
 if (command.equals("m3") ) {
    gui.datasea.mag(words, num_words, "both", 3, "+");
```

```
            }
        else
         if (command.equals("m4") ) {
            gui.datasea.mag(words, num_words, "both", 4, "+");
            }
        else
         if (command.equals("m5") ) {
            gui.datasea.mag(words, num_words, "both", 5, "+");
            }
        else
         if (command.equals("m6") ) {
            gui.datasea.mag(words, num_words, "both", 6, "+");
            }
        else
         if (command.equals("m7") ) {
            gui.datasea.mag(words, num_words, "both", 7, "+");
            }
        else
         if (command.equals("magdown") ) {
            gui.datasea.magdownstream(words, num_words);
            }
        else
         if (command.equals("mag") ) {
            gui.datasea.mag(words, num_words, "both", 3, "+");
            }
        else
         if (command.equals("choices") ) { // print ANs > BIG_MAG for each dist
from Thes'
            gui.datasea.choices();
            }
        else
         if (command.equals("magtype") ) { // mag all distally to infinity
            gui.datasea.magtype(gui.lastNode, (String) null, 'd');
            gui.datasea.magtype(gui.lastNode, (String) null, 'p');
            }
        else
         if (command.equals("magd") ) { // mag all distally to infinity
            gui.datasea.magd();
            }
        else
         if (command.equals("Mag") ) {
            gui.datasea.mag(words, num_words, "both", 6, "+");
            }
        else
         if (command.equals("d1") ) {
            gui.datasea.mag(words, num_words, "both", 1, "-");
            }
        else
         if (command.equals("d2") ) {
            gui.datasea.mag(words, num_words, "both", 2, "-");
```

```
   }
else
 if (command.equals("d3") ) {
   gui.datasea.mag(words, num_words, "both", 3, "-");
   }
else
 if (command.equals("d4") ) {
   gui.datasea.mag(words, num_words, "both", 4, "-");
   }
else
 if (command.equals("d5") ) {
   gui.datasea.mag(words, num_words, "both", 5, "-");
   }
else
 if (command.equals("d6") ) {
   gui.datasea.mag(words, num_words, "both", 6, "-");
   }
else
 if (command.equals("d7") ) {
   gui.datasea.mag(words, num_words, "both", 7, "-");
   }
else
 if (command.equals("dec") ) {
    gui.datasea.mag(words, num_words, "both", 3, "-");
   }
else
 if (command.equals("Dec") ) {
    gui.datasea.mag(words, num_words, "both", 5, "-");
   }
else
 if (command.equals("input") ) {
   ret_node = words_input(words, input_string, num_words);
   }
else
 if (command.equals("zoom") ) {
   gui.datasea.word_zoom(words, num_words);
    gui.magscale *= 3.0;
       gui.auto_rescale = false;
       gui.status("auto_rescale = "+gui.auto_rescale);
   }
else
if (command.equals("a") ) {
       gui.spread_factor -= 0.05;
       GUI.P(0,"word_selector","spread_factor = "+gui.spread_factor);
       }
else
if (command.equals("A") ) {
       gui.spread_factor += 0.05;
       GUI.P(0,"word_selector","spread_factor = "+gui.spread_factor);
       }
```

```
else
if (command.equals("d") ) {
    gui.datasea.word_dump(words, num_words);
    }
else
if (command.equals("D") ) {
    gui.datasea.word_dump(words, num_words);
    }
else
if (command.equals("tree") ) {
   gui.print_tree(words, num_words);
    }
else
if (command.equals("printup") ) {
    gui.datasea.print_upstream((Node)null, gui.lastNode);
    }
else
if (command.equals("print") ) { // print results of all
    gui.datasea.print();
    }
else
if (command.equals("oldprint") ) {
    gui.datasea.word_print(words, num_words);
    }
else
if (command.equals("state") ) {
    gui.state();
    }
else
if (command.equals("halt") ) {
    gui.request_stop_thread();
    }
else
if (command.equals("run") ) {
    gui.run_thread();
    }
else
if (command.equals("stop") ) {
    gui.stop_thread();
    }
else
 if (command.equals("scale") ) {
        gui.auto_rescale = !gui.auto_rescale;
        GUI.P(0,"word_selector","auto_rescale = "+gui.auto_rescale);
        gui.status("auto_rescale = "+gui.auto_rescale);
    }
else
 if (command.equals("in") ) {
     gui.magscale *= 2.0;
        gui.auto_rescale = false;
```

```
            gui.status("auto_rescale = "+gui.auto_rescale);
        }
    else
    if (command.equals("In") ) {
        gui.magscale *= 6.0;
            gui.auto_rescale = false;
            gui.status("auto_rescale = "+gui.auto_rescale);
        }
    else
    if (command.equals("out")) {
        gui.magscale /= 2.0;
            gui.auto_rescale = false;
            gui.status("auto_rescale = "+gui.auto_rescale);
        }
    else
    if (command.equals("Out")) {
        gui.magscale /= 6.0;
            gui.auto_rescale = false;
            gui.status("auto_rescale = "+gui.auto_rescale);
        }
    else
    if (command.equals("up")) {
        gui.WindowYOffset += 150/gui.magscale;
            gui.auto_rescale = false;
            gui.status("auto_rescale = "+gui.auto_rescale);
        }
    else
    if (command.equals("Up")) {
        gui.WindowYOffset += 450/gui.magscale;
            gui.auto_rescale = false;
            gui.status("auto_rescale = "+gui.auto_rescale);
        }
    else
    if (command.equals("do") ) {
        gui.WindowYOffset -= 150/gui.magscale;
            gui.auto_rescale = false;
            gui.status("auto_rescale = "+gui.auto_rescale);
        }
    else
    if (command.equals("Do")) {
        gui.WindowYOffset -= 450/gui.magscale;
            gui.auto_rescale = false;
            gui.status("auto_rescale = "+gui.auto_rescale);
        }
    else
    if (command.equals("le")) {
        gui.WindowXOffset -= 150/gui.magscale;
            gui.auto_rescale = false;
            gui.status("auto_rescale = "+gui.auto_rescale);
        }
        else
        if (command.equals("Le")) {
            gui.WindowXOffset -= 450/gui.magscale;
                gui.auto_rescale = false;
                gui.status("auto_rescale = "+gui.auto_rescale);
            }
        else
        if (command.equals("ri")) {
            gui.WindowXOffset += 150/gui.magscale;
                gui.auto_rescale = false;
                gui.status("auto_rescale = "+gui.auto_rescale);
            }
        else
        if (command.equals("Ri")) {
            gui.WindowXOffset += 450/gui.magscale;
                gui.auto_rescale = false;
                gui.status("auto_rescale = "+gui.auto_rescale);
            }
        else
         if (command.equals("s") ) {
             gui.spread_factor *= 0.9;
            }
        else
         if (command.equals("S") ) {
             gui.spread_factor *= 1.1;
            }
        else
         if (command.equals("q") ) {
             gui.quit();
            }
        else
         if (command.equals("m") || command.equals("more")) {
                gui.datasea.moreless("m");
            }
        else
         if (command.equals("l") || command.equals("less")) {
                gui.datasea.moreless("l");
            }
        else
         if (command.equals("t") ) {
                gui.mode_obj.toggle_draw_text();
            }
/***********************************************************************
        else
         if (command.equals("settiny") ) {
                gui.TinyNode = gui.lastNode;
                if (gui.TinyNode != null) {
                        if (gui.TinyNode.TinyScale == 1.0)
                                gui.TinyNode.TinyScale = 0.2;
                        else
```

```
                                gui.TinyNode.TinyScale = 1.0;
                        GUI.P(0,"word_selector","TinyNode "+gui.TinyNode.Name+"
Scale="+
                                gui.TinyNode.TinyScale);
                        }
                }
        else
         if (command.equals("maketiny") ) {
                gui.TinyNode = gui.lastNode;
                if (gui.TinyNode != null) {
                        if (gui.TinyNode.TinyScale == 1.0)
                                gui.TinyNode.TinyScale = 0.2;
                        else
                                gui.TinyNode.TinyScale = 1.0;
                        GUI.P(0,"word_selector","TinyNode "+gui.TinyNode.Name+"
Scale="+
                                gui.TinyNode.TinyScale);
                        gui.TinyFlag = true;
                        }
                }
        else
         if (command.equals("tiny") ) {
                gui.TinyFlag = !gui.TinyFlag;
                GUI.P(0,"word_selector","TinyFlag is "+gui.TinyFlag);
            }
        else
         if (command.equals("details")) {
                gui.Details = !gui.Details;
                gui.status("Details = "+gui.Details);
            }
*************************************************************************/
        else
         if (command.equals("update") ) {
                gui.datasea.needdistUpdate = true;
            }
        else
         if (command.equals("parse")) {
                gui.parse = !gui.parse;
                GUI.P(0,"word_selector","parse is "+gui.parse);
                gui.status("parse is "+gui.parse);
            }
        else
         if (command.equals("debug")) {
                if (gui.lastNode != null) {
                gui.lastNode.Debug = !gui.lastNode.Debug;
                GUI.P(0,"word_selector","lastNode("+gui.lastNode.Name+").Debug is
"+gui.lastNode.Debug);
                }
            }
        else
```

```
 if (command.equals("diag")) {
        gui.diag_frame.setVisible(true);
    }
else
 if (command.equals("nodiag")) {
        gui.diag_frame.setVisible(false);
    }
else
 if (command.equals("u") ) {
        gui.datasea.undo(1);
    }
else
 if (command.equals("smp") ) {
        gui.datasea.simplify(words, num_words);
    }
else
 if (command.equals("trim") ) { // m=minus
        gui.datasea.vote_branch(words, num_words, "-");
    }
else
 if (command.equals("grow") ) { // p=plus
        gui.datasea.vote_branch(words, num_words, "+");
    }
else
 if (command.equals("yes") ) { // p=plus
        gui.datasea.vote_branch(words, num_words, "+");
    }
else
 if (command.equals("no") ) { // m=minus
        gui.datasea.vote_branch(words, num_words, "-");
    }
else
 if (command.equals("boxes") ) {
        gui.drawBoxes = !gui.drawBoxes;
        GUI.P(0,"word_selector","drawBoxes is "+gui.drawBoxes);
    }
else
 if (command.equals("4") ) {
        gui.datasea.reset_mags(Node.MAX_MAG);
    }
else
 if (command.equals("3") ) {
        gui.datasea.reset_mags(Node.BIG_MAG);
    }
else
 if (command.equals("2") ) {
        gui.datasea.reset_mags(Node.MED_MAG);
    }
else
 if (command.equals("1") ) {
```

```
            gui.datasea.reset_mags(Node.BARELY_VISIBLE_MAG+.2);
        }
    else
    if (command.equals("0") ) {
            gui.datasea.reset_mags(Node.BARELY_INVISIBLE_MAG);
        }
    else
    if (command.equals("x") ) {
            gui.coordinates_on = ! gui.coordinates_on;
        }
    else
    if (command.equals("fla") ) {
            gui.datasea.flatten();
        }
    else
    if (command.equals("sha") ) {
            gui.datasea.sharpen();
        }
    else
    if (command.equals("norm") ) {
            gui.datasea.normalize();
        }
    else
    if (command.equals("newframe") ) {
            gui.extra_frame();
        }
    else
    if (command.equals("test") ) {
            gui.datasea.test();
        }
    else
    if (command.equals("stopspread") ) {
            if (gui.lastNode != null) {
                    gui.lastNode.StopSpread = true;
                    gui.P(0,"input","StopSpread true for "+gui.lastNode.Name);
                    }
        }
    else
    if (command.equals("isolate") ) {
            gui.datasea.isolate();
        }
    else
    if (command.equals("clean") ) {
            gui.datasea.clean();
        }
    else
    if (command.equals("like") ) {
            gui.datasea.like(gui.lastNode);
        }
    else
```

```
            if (command.equals("inh") ) {
                    gui.datasea.inhibit(words, num_words);
                }
            else
            if (command.equals("inhd") ) { // inh all distally to infinity
                gui.datasea.inhd();
                }
            else
            if (command.equals("drill") ) {
                    gui.datasea.drill(words, num_words);
                }
            else
            if (command.equals("set") ) {
                    gui.datasea.parse_set_cmd(words, num_words);
                }
            else
            if (command.equals("recent") ) {
                    gui.datasea.recent();
                }
            else
            if (command.equals("lm") ) {
                    gui.mode_obj.toggle_line_mode();
                }
            else
            if (command.equals(".") ) {
                    gui.datasea.repeat_priorCommand();
                }
            else
            if (command.equals("rm") ) {
                    gui.mode_obj.toggle_render_mode();
                }
            else
            if (command.equals("pm") ) {
                    gui.mode_obj.toggle_position_mode();
                }
            else
            if (command.equals("sm") ) {
                    gui.mode_obj.toggle_spread_mode();
                }
            else
           if (command.equals("auto")) {
                gui.datasea.auto_flatten();
                }
            else
           if (command.equals("pol")) {
                    gui.checkPolarization = !gui.checkPolarization;
                    GUI.P(0,"word_selector", "gui.checkPolarization =
"+gui.checkPolarization);
                }
            else
```

```
        if (command.equals("clear")) {
            gui.datasea.init();
            }
        else
        if (command.equals("clearstatus")) {
            gui.clear_status();
            }
        else
        if (command.equals("clearmsg")) {
            gui.global_str[0] = "";
            gui.global_str_size = 0;
            }
        else
        if (command.equals("action")) {
             gui.action(gui.lastNode);
            }
        else
         if (command.equals("get") ) {
                if (null != gui.lastNode)
                        GUI.P(0,"word_selector", "getting: "+gui.lastNode.Name);
                        gui.datasea.pop.get_a_URL(gui.lastNode);
                }
        else
         if (command.equals("readfile") ) {
                gui.datasea.pop.read_in_net_file(0);
                }
        else
         if (command.equals("readfile1") ) {
                gui.datasea.pop.read_in_net_file(1);
                }
        else
         if (command.equals("readfile2") ) {
                gui.datasea.pop.read_in_net_file(2);
                }
        else
         if (command.equals("getfile") ) {
                gui.datasea.pop.get_file();
                }
        else
         if (command.equals("tallis") ) {
                gui.datasea.pop.pull_in_URLs((String)"www.Tallis.com/",
"TOC.html", 1, (Node)null);
                }
        else
         if (command.equals("newtest") ) {
                gui.datasea.newtest();
                }
        else
         if (command.equals("poprev") ) {
                gui.datasea.pop.poprev();
                }
        else
         if (command.equals("popegg") ) {
                gui.datasea.pop.popegg();
                }
        else
         if (command.equals("poptal") ) {

        gui.datasea.pop.pull_in_URLs((String)"file:/usr/people/rocky/CIM/web/Tall
isBack/CIM.html", "", 1, (Node)null);
                }
        else
         if (command.equals("popCIM") ) {

        gui.datasea.pop.pull_in_URLs((String)"file:/usr/people/rocky/CIM/web/",
"index.html", 1, (Node)null);
                }
        else
         if (command.equals("poptallis") ) {
                gui.datasea.pop.pull_in_URLs((String)"http://www.Tallis.com/",
"Profile.html", 1, (Node)null);
                }
        else
         if (command.equals("popucb") ) {
                gui.datasea.pop.pull_in_URLs((String)"http://www.berkeley.edu/",
"index.html", 1, (Node)null);
                }

        else
         if (command.equals("showdist") ) {
                gui.datasea.showdist(words, num_words);
                }

        else
         if (command.equals("run1") ) {
                gui.run1();
                }
// populate POPULATE
        else
         if (command.equals("popframe") ) {
                gui.datasea.pop.popframe();
                gui.datasea.needdistUpdate = true;
                }
        else
         if (command.equals("popBB") ) {
                gui.datasea.pop.popBB();
                gui.datasea.needdistUpdate = true;
                }
        else
         if (command.equals("popfiles") ) {
                gui.datasea.pop.pull_in_URLs((String)null, (String)null, 1,
(Node)null);
                gui.datasea.needdistUpdate = true;
                }
        else
         if (command.equals("pop1") ) {
                gui.datasea.pop.pop_initial();
                gui.datasea.pop.popd();
                gui.datasea.pop.pops();
                gui.datasea.pop.popm();
                gui.datasea.pop.popf();
                gui.datasea.pop.popn();
                gui.datasea.pop.popx();
                //gui.datasea.pop.pop_properties();
                gui.datasea.needdistUpdate = true;
            }
        else
         if (command.equals("pop2") ) {
                gui.datasea.pop.popEcon();
                gui.datasea.pop.popDM();
                gui.datasea.pop.popSemi();
                gui.datasea.pop.popURLs();
                gui.datasea.pop.popfab();
                gui.datasea.pop.popmap();
                gui.datasea.needdistUpdate = true;
            }
        else
         if (command.equals("popgroc") ) {
                gui.datasea.pop.popgroc();
                gui.datasea.needdistUpdate = true;
            }
        else
         if (command.equals("popcntl") ) {
                gui.datasea.pop.popcntl();
                gui.datasea.needdistUpdate = true;
            }
        else
        if (command.equals("popapps") ) {
                gui.datasea.pop.popapps();
                gui.datasea.needdistUpdate = true;
           }
//      else
//         if (command.equals("popportal") ) {
//              gui.datasea.pop.popportal();
//            }
        else
         if (command.equals("popp") ) {
                gui.datasea.pop.popp();
                gui.datasea.needdistUpdate = true;
            }
        else
         if (command.equals("poppp") ) {
                gui.datasea.pop.poppp();
                gui.datasea.needdistUpdate = true;
            }
        else
         if (command.equals("popnet") ) {
                gui.datasea.pop.popnet();
                gui.datasea.needdistUpdate = true;
            }
        else
         if (command.equals("popw") ) {
                gui.datasea.pop.popweb();
                gui.datasea.needdistUpdate = true;
            }
        else
         if (command.equals("popinitial") ) {
                gui.datasea.pop.pop_initial();
                gui.datasea.needdistUpdate = true;
            }
//            if (command.equals("popin") ) {
//                gui.datasea.pop.popin();
//            }
        else
        if (command.equals("popsynonyms")) {
             gui.datasea.pop.pop_synonyms();
                gui.datasea.needdistUpdate = true;
            }
        else
         if (command.equals("popEcon") ) {
                gui.datasea.pop.popEcon();
                gui.datasea.needdistUpdate = true;
            }
        else
         if (command.equals("popDM") ) {
                gui.datasea.pop.popDM();
                gui.datasea.needdistUpdate = true;
            }
        else
        if (command.equals("popSemi")) {
             gui.datasea.pop.popSemi();
                gui.datasea.needdistUpdate = true;
            }
        else
        if (command.equals("popURLs")) {
             gui.datasea.pop.popURLs();
                gui.datasea.needdistUpdate = true;
            }
        else
        if (command.equals("popt")) {
            gui.datasea.pop.popt();
               gui.datasea.needdistUpdate = true;
          }
      else
      if (command.equals("popTL")) {
            gui.datasea.pop.popTL();
               gui.datasea.needdistUpdate = true;
          }
      else
      if (command.equals("popd")) {
            gui.datasea.pop.popd();
               gui.datasea.needdistUpdate = true;
          }
      else
      if (command.equals("demo1")) {
           gui.demo1();
          }
      else
      if (command.equals("popm")) {
            gui.datasea.pop.popm();
               gui.datasea.needdistUpdate = true;
          }
      else
      if (command.equals("demo2")) {
           gui.demo2();
          }
      else
      if (command.equals("rem")) {
            gui.datasea.remember();
          }
      else
      if (command.equals("for")) {
            gui.datasea.forget();
          }
      else
      if (command.equals("peg")) {
            gui.datasea.peg();
          }
      else
      if (command.equals("pops")) {
            gui.datasea.pop.pops();
          }
      else
      if (command.equals("popf")) {
            gui.datasea.pop.popf();
          }
      else
      if (command.equals("demo3")) {
           gui.demo3();
          }
```

```
        else
        if (command.equals("popx")) {
             gui.datasea.pop.popx();
                gui.datasea.needdistUpdate = true;
            }
        else
        if (command.equals("popn")) {
             gui.datasea.pop.popn();
                gui.datasea.needdistUpdate = true;
            }
        else
        if (command.equals("popmap")) {
             gui.datasea.pop.popmap();
                gui.datasea.needdistUpdate = true;
            }
        else
        if (command.equals("popfab")) {
             gui.datasea.pop.popfab();
                gui.datasea.needdistUpdate = true;
            }
        else
        if (command.equals("popcal")) {
             gui.datasea.pop.popcalendar();
                gui.datasea.needdistUpdate = true;
            }
        else
        if (command.equals("popc")) {
             gui.datasea.pop.popc();
                gui.datasea.needdistUpdate = true;
            }
        else
       if (command.equals("why")) {
            gui.datasea.word_trace(words, num_words);
            }
        else
       if (command.equals("trace")) {
            gui.datasea.word_trace(words, num_words);
            }
        else
        if (command.equals("whats")) {
            gui.datasea.whats(words, num_words);
            }
        else
        understood = false;
        if (!understood)
                GUI.WARNING(0,"word_selector","Command `"+words[0]+"' not
understood.");

        //gui.datasea.needdistUpdate = true;
```

```
return(ret_node);
} // end word_selector



/**
 ** method word_mode_command     set the spread mode
*/
        public void word_mode_command (String[] words, int num_words) {
        if (num_words == 0) {
                GUI.ERROR(0,"word_mode_command","Zero words given.");
                return;
                }
        GUI.P(0,"word_mode_command","Got command '" +words[0]+"'");
//
        if (words[0].equals("sr"))
                gui.mode_obj.set_spread_mode("radial");
        else if (words[0].equals("sd"))
                gui.mode_obj.set_spread_mode("distal");
        else if (words[0].equals("sp"))
                gui.mode_obj.set_spread_mode("proximal");
//
/************************
        else if (words[0].equals("lr"))
                gui.mode_obj.toggle_lines_mode("radial");
        else if (words[0].equals("ld"))
                gui.mode_obj.toggle_lines_mode("distal");
        else if (words[0].equals("lp"))
                gui.mode_obj.toggle_lines_mode("proximal");
**************/
//
        else if (words[0].equals("snap"))
                gui.mode_obj.set_render_mode("VR");
        else if (words[0].equals("nosnap"))
                gui.mode_obj.set_render_mode("relations");
//
        else if (words[0].equals("pr"))
                gui.mode_obj.set_position_mode("relations");
        else if (words[0].equals("pl"))
                gui.mode_obj.set_position_mode("levels");

        else
                GUI.ERROR(0,"word_mode_command","Unknown command '"
                        +words[0]+"'");


} // end word_mode_command
```

```
/**
 ** method words_input
*/
        public  Node words_input (String[] passed_words, String input_string, int
num_words) {
        Node note_node=null;

// put the string into a note_node without the first word 'input'

String s = "";
if (num_words > 1)
        s = passed_words[1];
for (int i=2;i<num_words;i++)
        s = s+" "+passed_words[i];

// check the CN 'Notes'
if (gui.datasea.Notes == null) {
        gui.datasea.Notes = gui.datasea.pop.create_node("Notes", "AN");
// HERE
        //gui.datasea.Notes.isCN = true;
        }

Node triplet_node = search_for_possessive(passed_words, note_node, note_node);
// ==>
if (triplet_node == null) {
        note_node = gui.datasea.pop.create_node(s, "DN");
        gui.datasea.Notes.link(note_node); // link the whole text node to 'Notes'
        check_times(passed_words, gui.datasea.Notes.Name, note_node);
        String[] Str2 = discard_words(passed_words);
        parse_and_link(Str2, note_node, note_node);
        }

return(note_node);

} // end words_input


/**
 **  check_times   look for time words and link given node to actual time
referred to
 **
*/
public void check_times (String[] passed_words, String CNode_name, Node node) {

int num_words = passed_words.length;
```

```
for (int i=0; i< num_words; i++) {
        //  Search for time words
        if (passed_words[i].equalsIgnoreCase("yesterday")) {
                gui.tl.create_TS(node, CNode_name, "Mar 11 2000");
//System.out.print("check_times: linking `Mar 11 2000' (yesterday) to
<"+node.Name+">");
                }
        else if (passed_words[i].equalsIgnoreCase("today")) {
                gui.tl.create_TS(node, CNode_name, "Mar 12 2000");
//System.out.print("check_times: linking `Mar 12 2000' (today) to
<"+node.Name+">");
                }
        else if (passed_words[i].equalsIgnoreCase("tomorrow")) {
                gui.tl.create_TS(node, CNode_name, "Mar 13 2000");
//System.out.print("check_times: linking `Mar 13 2000' (tomorrow) to
<"+node.Name+">");
                }
        else if (passed_words[i].equalsIgnoreCase("1999")) {
                gui.tl.create_TS(node, CNode_name,
array_to_string(passed_words));
//System.out.print("check_times: checking times for <"+node.Name+">");
                }
        else if (passed_words[i].equalsIgnoreCase("2000")) {
                gui.tl.create_TS(node, CNode_name,
array_to_string(passed_words));
//System.out.print("check_times: checking times for <"+node.Name+">");
                }
        } // end for i

} // end check_times


/**
 **  search_for_possessive
 **
*/
public Node search_for_possessive (String[] passed_words, Node central_node,
Node CNode) {
int i, index, num_words=0;
Node child;
Node ret_node=null;


num_words = passed_words.length;

// Search for possesive "'s" sign, create AN-DN combination for the possesive
for (i=0; i<num_words; i++) {
        if (0 < (index = passed_words[i].indexOf("'s"))) {
        //ret_val = true;
        GUI.P(1, "search_for_possessive","num_words="+num_words+", i="+i);
        GUI.P(1, "search_for_possessive","passed_words[i]="+passed_words[i]);
        if (num_words < (i+1+3)) {
                GUI.WARNING(0, "search_for_possessive","Too few words");
                } else {
                String word1 = passed_words[i].substring(0,index); // BOB's hair
is red
                String word2 = passed_words[i+1]; // bob's HAIR is red
                String word3 = passed_words[i+3]; // bob's hair is RED
                String desc = "is";
                if (passed_words[i+2].equalsIgnoreCase("is")) {
                        if (num_words>=(i+1+4)) // see if there are enough words
to check
                                if (passed_words[i+3].equalsIgnoreCase("a")) {
                                        desc = "is a"; // change description
                                        word3 = passed_words[i+4]; // correct word3
                                        }
                        }
                // w1 <-> w3 <-> w2
                GUI.P(1,"search_for_possessive",
                "POSSESSIVE: word1="+word1
                +" <--> word2="+word2
                +" <--> desc="+desc
                +" <--> word3="+word3);
                ret_node = gui.datasea.pop.triplet(word1, word3, word2);
/***************************************************************
                context_node.link(gui.datasea.cl(word1));
                context_node.link(gui.datasea.cl(word2));
                context_node.link(gui.datasea.cl(word3));

                Node w2_node=null;
                if (null != (w2_node=gui.datasea.find_node_named(word2,"AN")))
                        w2_node.set_Desc(gui.datasea.find_node_named(word3,"DN"),
desc);
***********************************************************/
                }
        }
}
return(ret_node);
} // end search_for_possessive


/**
 **  parse_and_link
 **
*/
public void parse_and_link (String[] passed_words, Node central_node, Node
CNode) {
int i, num_words;
Node child, word_node=null;
```

```
num_words = passed_words.length;

// LINK INDIVIDUAL WORDS TO PASSED CNode
for (i=0; i< num_words; i++) {
       if (passed_words.equals("")) // blank words have been 'discarded'
previously
                     ;
////////////////////////////////////////
////////////////////////////////////////
       else {
//System.out.print("parse_and_link: linking <"+passed_words[i]+"> to
<"+central_node.Name+">");
       if (null == (word_node=gui.datasea.find_node_named(passed_words[i])))
              word_node = gui.datasea.pop.create_node(passed_words[i], "DN");
       central_node.link(word_node);
       //central_node.link(word_node, CNode);  // this makes the Note node a
CNode
       //gui.datasea.addCNodeBetweenNodes(central_node, word_node, CNode);
       }
       } // end for i

} // end parse_and_link


/**
 **  is_useless_word
 **
*/
public boolean is_useless_word (String word) {
int i, size;
Node child;

if (word == null) {
       GUI.WARNING(0,"is_useless_word","null word.");
       return(true);
       }


String lower_case_word = word.toLowerCase();

if (0 <= gui.datasea.useless_words.indexOf(lower_case_word) ) {
       //System.out.println("Useless: "+lower_case_word);
       return (true);
       }
else
       return(false);

} // end is_useless_word

/**
```

```
 **  discard_words   clobber useless words
 **
*/
public String[] discard_words (String[] passed_words) {
int i, j=0, num_words;
String[] ret_words;

num_words = passed_words.length;

String[] good_words = new String[num_words];

for (i=0; i< num_words; i++) {
       if (is_useless_word(passed_words[i])) { // Search for useless words
       }
       else
       {
       good_words[j++] = passed_words[i];
       //System.out.print(passed_words[i]+" ");
       }
} // end for i


// THIS IS STUPID, BUT I WANT TO RETURN AN ARRAY OF ONLY GOOD WORDS
ret_words = new String[j];
for (i=0; i<j; i++)
       ret_words[i] = good_words[i];
return(ret_words);
} // end discard_words

/**
 **  discard_words   String input and output version
 **
*/
public String discard_words (String s) {
int i;

String[] words = string_to_array(s);

// CALL THE STRING[] VERSION OF OURSELF
String[] ret_words = discard_words(words);

// REFORMAT BACK TO ORIGINAL FORM
String ret_string = array_to_string(ret_words); // when do we have to say 'new'?

return(ret_string);
} // end string version of discard_words


/**
 **  array_to_string
 **
*/
public String array_to_string (String[] str_array) {
int i;

// PUT BACK INTO STRING
String ret_string=""; // when do we have to say 'new'?
for (i=0; i<str_array.length; i++)
        ret_string = ret_string+str_array[i]+" ";
return(ret_string);
} // end array_to_string


/**
 **  array_to_string
 **
*/
public String[] string_to_array (String s) {
int i;

// TOKENIZE
StringTokenizer st = new StringTokenizer(s, " ,.?<>\"\t\r\n" );
int num_words = st.countTokens();
String[] words = new String[num_words];
for (i=0; i<num_words; i++)
        words[i] = st.nextToken();
return(words);
} // end array_to_string


/**
 **  parse_data
 **
*/
public void parse_data (Node node) {
int i, j, size, num_words=0, index=0, counter=0;
Node child;
String words[];
String line;

if (!gui.parse) {
        return;
        }

if (node==null) {
        return;
        }
if (node.Data==null) {
        return;
        }
if (node.Data[0]==null) {
        return;
        }

for (i=0; i<Node.MAX_TEXT_DATA_LINES; i++) {
        line = node.Data[i];
        if (line == null)
               break;

        line = gui.eliminate_html(line);
        words = string_to_array(discard_words(line));

        num_words = words.length;
         for (j = 0; j < num_words; j++) {
               child = gui.datasea.pop.create_node(words[j], "DN");
               //System.out.println(node.Name+", "+words[j]);
               node.link(child, "");
               }
        }

return;
} // end parse_data


} // end Input
```

```
// This is Populate.java     by Rocky Nevin

import java.util.*;
import java.io.*;
import java.awt.*;

// POPULATING and CREATING NODES


public class Populate extends Object {
DataSea datasea;
static Node lastBBnode; // a bad way, but, nonetheless, used by make_BB()
static Node AllURLs = null;
static Node Cluster_Files;
static Node Cluster_Web;
static Node URLs;
static Node Cluster_Mail;
static Node Cluster_Directory;
static Node Cluster_Notes;
static Node Cluster_En;
static Node TimeLine;
static Node yesterday_node, today_node, tomorrow_node;
static Node last_week_node, this_week_node, next_week_node;
static int MAX_FILES_PER_DIRECTORY=15;
static int MAX_DIRECTORY_DEPTH=4;
static int global_counter=0;
static int counter=0;
static int calls_to_pull=0;
//static int file_counter=0;
GetURLInfo url;

public Populate (DataSea passed_datasea_obj) {       // Constructor
datasea = passed_datasea_obj;
this.init();
} // end Populate constructor


public void init () {
}

public void populate_begin () {
GUI.sleep(100);
System.err.println("Populate.populate_begin() begun...");
//GUI.sleep(100);
        pop_initial();
        read_in_net_file(0);
//GUI.sleep(100);
//GUI.sleep(100);
//      popportal();
//GUI.sleep(100);
```

```
        popd();
//GUI.sleep(100);
        popgeneology();
//GUI.sleep(100);
        popp();
//GUI.sleep(100);
        popTL();
//GUI.sleep(100);
        popcalendar();
//GUI.sleep(100);
        popn();
System.err.println("Populate.populate_begin() done.");
} // end populate_begin



/**
 **  get_file
 **
*/
public void get_file () {
GetURLInfo url;
byte byte_array[]=new byte[300];
int i, index;
String w1=null, w2=null;
int max_lines = 700;
FileInputStream file;

try { file = new FileInputStream("/usr/people/rocky/x"); }
catch (java.io.FileNotFoundException e)
        {
        GUI.WARNING(0,"get_file"," ->"+e.toString()+"<- ");
        return;
        }
try { file.read(byte_array); }
catch (java.io.IOException e)
        {
        GUI.WARNING(0,"get_file"," ->"+e.toString()+"<- ");
        return;
        }
System.out.println("byte_array from /usr/people/rocky/x:");
System.out.println(byte_array);

} //get_file

/**
 **  read_in_net_file
 **
*/
```

```
public void read_in_net_file(int which) {
GetURLInfo url;
String str_array[]=new String[0];
int i, index;
String w1=null, w2=null, name, value;
int max_lines = 700;
String FileName = "/README";

if (which == 1)
       FileName = "/README.1";
else
if (which == 2)
       FileName = "/README.2";

url = new GetURLInfo();
try { str_array = url.get_URL("file:"+GUI.GlobalUserDir+FileName, max_lines); }
catch (java.net.MalformedURLException e)
       {
       GUI.WARNING(0,"read_in_net_file"," ->"+e.toString()+"<- ");
       return;
       }
catch (IOException e)
       {
       GUI.WARNING(0,"read_in_net_file"," ->"+e.toString()+"<- ");
       return;
       }

if (str_array == null) {
       GUI.WARNING(0,"read_in_net_file"," str_array is null.");
       return;
       }

DataSea.currentCNode = create_node("ThesCN", "CN"); // Turn on auto-CNode links

for (i=0; i<max_lines; i++) {

       if (str_array[i] == null)
              break;

if (str_array[i].length() > 0)
if (!str_array[i].substring(0,1).equals("#")) {

index = str_array[i].indexOf(",");
if (index <= 0)
       break;
w1 = str_array[i].substring(0,index);
w2 = str_array[i].substring(index+2);

//meta subject:subject_value
if (0<=w1.indexOf("meta")) {
        index = w1.indexOf(":");
        name = w1.substring(5, index);
        value = w1.substring(index+1);
        METApair(name, value, w2);
        }
else
if (0<=w1.indexOf("http"))
        if (0<=w2.indexOf("http"))
                URLtoURLpair(w1, w2);
        else
                URLtoANpair(w1, w2);
else
        ANANpair(w1, w2);


} // if not '#'
} // for i
System.err.println(" Total lines: "+i);

DataSea.currentCNode = null; // Turn off auto-CNode links

return;
} // end get_file_input




/**
 **  pull_in_URLs
 **
*/
public void pull_in_URLs (String base, String name, int current_depth, Node
parent) {
String str_array[]=new String[0];
String url_name="UNKNOWN";
String domain_name=null;
String line;
int i;
int index=0, end=0, end_space, end_bracket;
int MAX_CHILDREN = 7-current_depth;
int MAX_DEPTH = 4; // set conditionally below on the OS
int child_count=0;
Node node=null, cn_node=null;

if (current_depth > MAX_DEPTH) {
        return;
        }
if (current_depth == 1)
        calls_to_pull = 0;
```

```
if (parent == null)  // starting, so link first to a reliable node
if (null == (parent = datasea.find_node_named("URLs"))) {
        (parent = create_node("URLs", "DN")).link(datasea.Root);
        }

if (AllURLs == null)
        AllURLs = create_node("AllURLs","DN"); // link all to this one

if ((System.getProperty("os.name")).equalsIgnoreCase("Irix")) {
        MAX_DEPTH = 3;
        if (base == null)
                base = "file:/../../CIM/web/";
        }
else {
        MAX_DEPTH = 6; // Using cable-modem on Wintel
        if (base == null)
                base = "file:./";
        }
        if (name == null)
                name = ""; // just get the files

if (calls_to_pull++ > 30)
        {
        datasea.gui.P(0, "pull_in_URLs","calls_to_pull = "+calls_to_pull);
        return;
        }


        url_name = base+name;

//System.err.println("1) STARTING parent       ->"+parent.Name+"<- ");
//System.err.println("1)           base is           ->"+base+"<-");
//System.err.println("1)                   name is           ->"+name+"<-");
if ((0<=url_name.toLowerCase().indexOf("http")) && !datasea.gui.NetOK) {
        datasea.gui.P(0, "pull_in_URLs",
        "Sensed http, and GUI.NetOK is false, so aborting data retrieval for
"+url_name);
        return;
        }

        i= 1 + url_name.lastIndexOf("/"); // recalculate the base and the file
name
        if (i>0) {
                base = url_name.substring(0,i);
                name = url_name.substring(i);
//System.err.println(parent.Name+", "+url_name);
//System.err.println("                    base is            ->"+base+"<-, name is -
>"+name+"<-");
//System.err.println("                            name is          ->"+name+"<-");
```

```
                }
            url = new GetURLInfo();
            try { str_array = url.get_URL(url_name); }
            catch (java.net.MalformedURLException e)
                {
                //GUI.WARNING(0,"pull_in_URLS"," ->"+e.toString()+"<- ");
                return;
                }
            catch (IOException e)
                {
                //GUI.WARNING(0,"pull_in_URLS"," ->"+e.toString()+"<- ");
                return;
                }
        if (str_array == null) {
            //GUI.WARNING(0,"pull_in_URLS"," str_array is null.");
            return;
            }



        // Create and link a node for this URL if it doesn't exist already
        if (null == (node = datasea.find_node_named(url_name))) {
            node = create_node(url_name, "URL");
            }

            parent.link(node);
            //GUI.P(0,"pull_in_URLS"," linking AllURLs to "+node.Name);
            AllURLs.link(node); // link all to this one

        //
        // figure out the domain name, use as a CN ...
        //
        if (null != (domain_name=datasea.gui.find_domain_name(url_name))) {
        if (null == (cn_node = datasea.find_node_named(domain_name))) {
            cn_node = create_node(domain_name, "CN");
            }
            cn_node.link(node);
            //AllURLs.link(cn_node);
            //GUI.P(0,"pull_in_URLS"," linking CN "+cn_node.Name);
            }
        else
            GUI.WARNING(0,"pull_in_URLS","Failed to find good domain: domain_name is
        "+domain_name);


        node.isSelected = true;
        datasea.gui.show_node_once(node);

        for (i=0; i<Node.MAX_TEXT_DATA_LINES; i++) {
            if (str_array[i] == null)
```

```
                break;
        node.Data[i]=str_array[i];
        //System.out.println("Adding to "+node.Name+":    "+str_array[i]);
}
GUI.input.parse_data(node);

if (child_count > MAX_CHILDREN) {
        System.err.print(".");
        node.isSelected = false;
        return;
        }

// Now, look for URLs to call
        String new_name="";
        ++current_depth;
        if (current_depth > MAX_DEPTH) {
                node.isSelected = false;
                return;
                }
         for (i = 0; i < Node.MAX_TEXT_DATA_LINES; i++) {
                line = str_array[i];
             if (line == null) break;
        new_name = datasea.gui.find_URL_name(line);
        if (new_name != null) {
                if (0<=new_name.toLowerCase().indexOf("http")) {
                        pull_in_URLs("", new_name, current_depth, node);
                        }
                else {
                        pull_in_URLs(base, new_name, current_depth, node);
                        }
                }
        }
node.isSelected = false;
datasea.gui.update(1);
} // end pull_in_URLs


/**
 **  pull_in_URLs  no argument version
 **
*/
public void pull_in_URLs () {
System.err.println("============ pull_in_URLs  ... Begun
...===============================");
        pull_in_URLs((String)null, (String)null, 1, (Node)null);
System.err.println("============ pull_in_URLs
Done.=======================================");
System.err.println("size of node_vec is "+datasea.node_vec.size());
} // end no arg' version of pull_in_URLs
```

```
/**
 **  popMyFamily
 **
*/
public void popMyFamily () {  // "show geneology"
Node grape_node, cereal_node, nuts_node, wheat_node;
//
// ANs: grape, cereal, nuts, red, green, wine, wheat, zinfandel, crunchy
// DNs: Grape Nuts, Wheat Germ, Shredded Wheat, Acacia Zinfandel, Green Grapes,
Red Grapes,
//
//

Node geneologyCN = create_node("geneology","AN");

triplet("Rocky", "Jane Elinore GEIGER", "Mother", geneologyCN, "polarized");
triplet("Rocky", "Harry Wardlaw Jr. NEVIN", "Father", geneologyCN, "polarized");
triplet("Jane Elinore GIEGER", "Elinore Lucille LINGARD", "Mother", geneologyCN,
"polarized");
triplet("Jane Elinore GIEGER", "Charles Towne GEIGER", "Father", geneologyCN,
"polarized");
triplet("Harry Wardlaw Jr. NEVIN", "Hazel Miller HAWKINS", "Mother",
geneologyCN, "polarized");
triplet("Harry Wardlaw Jr. NEVIN", "Harry Wardlaw Sr. NEVIN", "Father",
geneologyCN, "polarized");
triplet("Elinore Lucille LINGARD", "Eliza Jane BAKER", "Mother", geneologyCN,
"polarized");

triplet("Elinore Lucille LINGARD", "Amos Lister LINGARD", "Father", geneologyCN,
"polarized");

triplet("Charles Towne GEIGER", "Grace TIDRICK", "Mother", geneologyCN,
"polarized");
triplet("Charles Towne GEIGER", "Eugene Warfel GEIGER", "Father", geneologyCN,
"polarized");

triplet("Harry Wardlaw Sr. NEVIN", "Lula Wardlaw", "Mother", geneologyCN,
"polarized");
triplet("Harry Wardlaw Sr. NEVIN", "Michael John NEVIN", "Father", geneologyCN,
"polarized");

triplet("Hazel Miller HAWKINS", "Janetta A CLOUGH", "Mother", geneologyCN,
"polarized");
triplet("Hazel Miller HAWKINS", "George Wicton HAWKINS", "Father", geneologyCN,
"polarized");

Node DirCNode = create_node("DirCN","CN");
triplet("Grace TIDRICK", "734 S.Main", "address", DirCNode, "polarized");
```

```
return;
} // end popMyFamily

/**
 **  popgeneology
 **
*/
public void popgeneology () {  // "show geneology"
Node grape_node, cereal_node, nuts_node, wheat_node;
//
// ANs: grape, cereal, nuts, red, green, wine, wheat, zinfandel, crunchy
// DNs: Grape Nuts, Wheat Germ, Shredded Wheat, Acacia Zinfandel, Green Grapes,
Red Grapes,
//
//
Node geneologyCN = create_node("geneology","AN");
Node employmentCN = create_node("employment","AN");
employmentCN.isCN = true;
geneologyCN.isCN = true;
Node peopleAN = create_node("people","AN");
DataSea.Root.link(geneologyCN);
DataSea.Root.link(employmentCN);
peopleAN.link(employmentCN, "polarized");
peopleAN.link(geneologyCN, "polarized");

triplet("Bob", "BobsDad", "Father", geneologyCN, "polarized");
triplet("Bob", "BobsSon", "Son", geneologyCN, "polarized");
triplet("BobsSon", "BobsGrandSon", "Son", geneologyCN, "polarized");

Node ann_node = create_node("Ann","DN"); //
Node ted_node = create_node("Ted","DN"); //
triplet("Bob", "Ann", "Wife", geneologyCN, "polarized");
triplet("Ann", "Ted", "Father", geneologyCN, "polarized");
triplet("Ted", "IBM", "employment", employmentCN, "polarized");

popMyFamily();

return;
} // end popgeneology

/**
 **  popgroc
 **
*/
public void popgroc () {  // "show grocery"
Node grape_node, cereal_node, nuts_node, wheat_node;
//
// ANs: grape, cereal, nuts, red, green, wine, wheat, zinfandel, crunchy
// DNs: Grape Nuts, Wheat Germ, Shredded Wheat, Acacia Zinfandel, Green Grapes,
Red Grapes,
//
//

Node grocery = create_node("grocery","DN");
DataSea.Root.link(grocery);
grape_node = create_node("grape","AN"); //
grocery.link(grape_node);
grape_node.link(datasea.cl("Grape Nuts","DN"));
grape_node.link(datasea.cl("Red Grape","DN"));
grape_node.link(datasea.cl("Green Grape","DN"));

wheat_node = create_node("wheat","AN"); //
grocery.link(wheat_node);
wheat_node.link(datasea.cl("Wheat Germ","DN"));
wheat_node.link(datasea.cl("White Flour","DN"));
wheat_node.link(datasea.cl("Shredded Wheat","DN"));

cereal_node = create_node("cereal","AN"); //
grocery.link(cereal_node);
cereal_node.link(datasea.cl("Grape Nuts","DN"));
cereal_node.link(datasea.cl("Shredded Wheat","DN"));
cereal_node.link(datasea.cl("Corn Flakes","DN"));



GUI.P(0, "popgroc", "show grocery");
return;
} // end popgroc

/**
 **  popcntl
 **
*/
public void popcntl () { // "show cntl"
Node file_node, edit_node, cntl_node, apps_node, pop_node;

// cntl - File - Open
//            - Save
//            - Export
//            - Import - File_1
//                     - File_2
//                     - File_3
//                     - File_4
//       - Edit - Cut
//              - Copy
//              - Paste
//              - Select All
//              - Delete
```

```
cntl_node = create_node("Cntl","AN"); // created also in popapps
apps_node = create_node("Apps","AN");
apps_node.link(cntl_node);
DataSea.Root.link(cntl_node);
cntl_node.link(datasea.cl("File","DN","Open","DN"));
cntl_node.link(datasea.cl("Edit","DN", "Cut","DN"));
file_node = datasea.find_node_named("File","DN");
edit_node = datasea.find_node_named("Edit","DN");
file_node.link(datasea.cl("Import", "DN", "File_1", "DN"));
(datasea.find_node_named("Import", "DN")).link(datasea.cl("File_2", "DN"));
(datasea.find_node_named("Import", "DN")).link(datasea.cl("File_3", "DN"));
(datasea.find_node_named("Import", "DN")).link(datasea.cl("File_4", "DN"));
file_node.link(datasea.cl("Export", "DN"));
file_node.link(datasea.cl("Save", "DN"));
file_node.link(datasea.cl("Save As", "DN"));
edit_node.link(datasea.cl("Copy", "DN"));
edit_node.link(datasea.cl("Paste", "DN"));
edit_node.link(datasea.cl("Select All", "DN"));
edit_node.link(datasea.cl("Delete", "DN"));

pop_node = create_node("Populate", "DN");
pop_node.link(create_node("popf","DN"));
pop_node.link(create_node("popw","DN"));
pop_node.link(create_node("popd","DN"));
pop_node.link(create_node("popURLs","DN"));
pop_node.link(create_node("popTL","DN"));
pop_node.link(create_node("popapps","DN"));

popapps();

GUI.P(0, "popcntl", "show cntl");
return;
} // end popcntl


/**
 **  popapps
 **
*/
public void popapps () { // "show apps"
Node mail_node, web_node, apps_node, cntl_node;

// Apps - Mail - Receive
//            - Send
//            - Cleanup
//            - Include File - Mail_1
//                           - Mail_2
//                           - Mail_3
//                           - Mail_4
//        - Web   - Open Browser
//                - Properties
//                - Review Bookmarks
//                - Home
//                - Blah

DataSea.currentCNode = create_node("AppsCN", "CN"); // Turn on auto-CNode links

cntl_node = create_node("Cntl","AN"); // created also in popmenu
apps_node = create_node("Apps","AN");
apps_node.link(cntl_node);
DataSea.Root.link(apps_node);
apps_node.link(datasea.cl("Mail","DN","Receive","DN"));
apps_node.link(datasea.cl("Web","DN", "Open Browser","DN"));
mail_node = create_node("Mail","DN");
web_node = create_node("Web","DN");
mail_node.link(datasea.cl("Include File", "DN", "Mail_1", "DN"));
(create_node("Include File", "DN")).link(datasea.cl("Mail_2", "DN"));
(create_node("Include File", "DN")).link(datasea.cl("Mail_3", "DN"));
(create_node("Include File", "DN")).link(datasea.cl("Mail_4", "DN"));
mail_node.link(datasea.cl("Cleanup", "DN"));
mail_node.link(datasea.cl("Save", "DN"));
mail_node.link(datasea.cl("Save As", "DN"));
web_node.link(datasea.cl("Properties", "DN"));
web_node.link(datasea.cl("Review Bookmarks", "DN"));
web_node.link(datasea.cl("Home", "DN"));
web_node.link(datasea.cl("Blah", "DN"));

GUI.P(0, "popapps", "show popapps");

DataSea.currentCNode = null; // Turn off auto-CNode links

return;
} // end popapps


/**
 **  properties
 **
*/
public void properties () {
Node prop_node;

GUI.P(0, "properties", "Creating Property nodes ('Props')");

prop_node = create_node("Props","DN","From System.getProperty() and
Toolkit.getDefaultToolkit().getScreenSize()");
DataSea.Root.link(prop_node);
```

```
triplet( "Props", System.getProperty("java.version"), "JavaVersion");
triplet( "Props", System.getProperty("java.vendor"), "JavaVendor");
triplet( "Props", System.getProperty("java.vendor.url"), "JavaVendorURL");
triplet( "Props", System.getProperty("java.class.version"), "JavaClassVersion");
triplet( "Props", System.getProperty("os.name"), "OS_Name");
triplet( "Props", System.getProperty("os.arch"), "OS_Arch");
triplet( "Props", System.getProperty("file.separator"), "FileSeparator");
triplet( "Props", System.getProperty("user.name"), "UserName");
triplet( "Props", System.getProperty("user.home"), "UserHome");
triplet( "Props", System.getProperty("user.dir"), "UserDir");
triplet( "Props", ""+Toolkit.getDefaultToolkit().getScreenSize(), "ScreenSize");


// GUI.GlobalUserDir = System.getProperty("user.dir");
// GUI.GlobalOSName = System.getProperty("os.name");

} // end properties



 /**
 **  pop_initial
 **
*/
        public void pop_initial () { // "show root"
         Node n1,n11,n12, n2,n21,n22, n33;
         Node Cntl;

GUI.P(0,"pop_initial", " begun ");

         if (DataSea.node_vec != null)
             DataSea.node_vec.removeAllElements();
         DataSea.node_vec = new Vector();

         GUI.selected_nodes_vec = new Vector(10);

         DataSea.Root = new Node("Root", "Layer", "", 0, 50, 10, 10);

         DataSea.Thesaurus_node = new Node("THES", "AN", "", 0, 0, 10, 10);
        TimeLine = new Node("TL", "Event", "TimeLine",
               -200, -100, 3, 400);
        //DataSea.Root.link(TimeLine);
        //(DataSea.Root.getLinkTo(TimeLine)).setLinksVRparms(TimeLine);

         Cntl = new Node("Cntl", "AN", "Control",
                         -300, 50, 3,3);
        Cluster_Files = new Node("Files", "AN", "Files",
                         -200, 50, 3,3);
        Cluster_Web = new Node("Web", "AN", "Web URLs",
                         -100, 50, 3,3);
        URLs = new Node("URLs", "CN", "URL CNode");

        Cluster_Mail = new Node("EMail", "AN", "E-mail",
                          0, 0, 3,3);
        Cluster_Directory = new Node("Dir", "AN", "Directory",
                          0, 0, 3,3);
        Cluster_Notes = new Node("Notes", "AN", "Free-form Notes",
                          0, 0, 3,3);
        Cluster_En = new Node("Encyclopedia", "AN", "Encyclopedia Online",
                        300, 50, 3,3);

//      Cluster_Mail.link( n1=create_node("Read-Mail", "AN"));
//      n1.link(create_node("file:/usr/people/rocky/Back/Code/Mail_DS/ltr-
1.html","URL"));
//      n1.link(create_node("file:/usr/people/rocky/Back/Code/Mail_DS/ltr-
2.html","URL"));
//      n1.link(create_node("file:/usr/people/rocky/Back/Code/Mail_DS/ltr-
3.html","URL"));

        Cluster_Web.link(create_node("http://www.metmuseum.org/htmlfile/faq/faq.h
tml", "URL"));


        DataSea.Root.StopSpread = true; // caught by recursive routines, block
spreading
        URLs.StopSpread = true;
        Cluster_Web.StopSpread = true;
        Cluster_Mail.StopSpread = true;
        Cluster_Notes.StopSpread = true;
        Cluster_En.StopSpread = true;
        Cntl.StopSpread = true;
        TimeLine.StopSpread = true;

        DataSea.Root.link(DataSea.Thesaurus_node);
        DataSea.Root.link(Cluster_Files);
        DataSea.Root.link(Cluster_Web);
        DataSea.Root.link(Cluster_Mail);
        DataSea.Root.link(Cluster_Notes);
        DataSea.Root.link(Cluster_Directory);
        DataSea.Root.link(Cluster_En);
        DataSea.Root.link(Cntl);
        DataSea.Root.setLinksVRparmsTo(Cluster_Files);
        //(DataSea.Root.getLinkTo(Cluster_Files)).setLinksVRparms(Cluster_Files);
        (DataSea.Root.getLinkTo(Cluster_Web)).setLinksVRparms(Cluster_Web);
        (DataSea.Root.getLinkTo(Cluster_Mail)).setLinksVRparms(Cluster_Mail);
        (DataSea.Root.getLinkTo(Cluster_Directory)).setLinksVRparms(Cluster_Direc
tory);
        (DataSea.Root.getLinkTo(Cluster_Notes)).setLinksVRparms(Cluster_Notes);
        (DataSea.Root.getLinkTo(Cluster_En)).setLinksVRparms(Cluster_En);
/********************************************
```

```
        Cntl.link(datasea.cl("Cmds", "AN", "", 0, 10));
        Cntl.link(datasea.cl("Graphics", "AN", "", 20, 30));
        Cntl.link(datasea.cl("Thresh", "AN", "", 40, 50));
        Cntl.link(datasea.cl("demos", "AN", "", 60, 70));
        Cntl.link(datasea.cl("Cmds", "AN", "", 80, 80));
***********************************************/

        n1= new Node("name", "AN");
        n11 = new Node("Rocky", "DN", "This is a test Node, named 'Rocky'");
        n1.link(n11);

        n12= new Node("Bob", "DN");
        n1.link(n12);

triplet( "CommitteeX", "Bob", "Member");

 datasea.needdistUpdate = true;
GUI.P(0,"pop_initial", " done. ");
} // end pop_initial


/**
 **  pop_synonyms
 **
*/
public void pop_synonyms () { // "show piano"

        GUI.input.string_input("input piano:syn:klavier");
        GUI.input.string_input("input phone:syn:telephone");
GUI.P(0, "pop_synonyms", "show piano");

} // end pop_synonyms


/**
 **  semi   Populate Semiconductor fab
 **
*/
        public void popSemi () { // skip
        Node Semi, t_node;


/**************************************************
        Semi =
(DataSea.Root.link(datasea.cl("SemiConductor","AN","",0,0)).link(datasea.cl("Sem
iConductor Mfg", "AN", "",0,0)));
                  Semi.link(datasea.cl("Semi Process","AN","",0,0));
        t_node = Semi.link(datasea.cl("www.semi.com","DN","URL",0,0));
        t_node = Semi.link(datasea.cl("URL-A","DN","URL",0,0));
        t_node = Semi.link(datasea.cl("URL-B","DN","URL",0,0));
        t_node = Semi.link(datasea.cl("URL-C","DN","URL",0,0));
        t_node = Semi.link(datasea.cl("PhD Thesis","DN","URL",0,0));
        t_node.link(datasea.find_node_named("URL-A"));
        t_node.link(datasea.find_node_named("URL-B"));
        t_node.link(datasea.find_node_named("URL-C"));
        t_node.link(datasea.find_node_named("www.semi.com"));
        (datasea.find_node_named("URL-A")).link(datasea.find_node_named("Semi
Process"));
***********************************************/
}// end popSemi


/**
 **  popDM   Populate Data Mining
 **
*/
        public void popDM () { // skip
         Node DM, tnode;
        int i;

GUI.P(0,"DM", "DataMining Run.");
/********************************************************
// Master node
        DM = DataSea.Root.link(datasea.cl("DM","AN","DataMining",-20,20));

// Gender node
        tnode = DM.link(datasea.cl("Gender", "AN", "Stats",-40,30));
                tnode.link(datasea.cl("Male", "DN", "Stats",-10,10));
                tnode.link(datasea.cl("Female", "DN", "Stats",10,10));
// Dept node
         tnode = DM.link(datasea.cl("Dept", "AN", "Stats",-20,30));
                tnode.link(datasea.cl("Sports", "DN", "",-10,10));
                tnode.link(datasea.cl("Clothing", "DN", "",0,10));
                tnode.link(datasea.cl("Appliances", "DN", "",10,10));
// Season node
        tnode = DM.link(datasea.cl("Season", "AN", "Stats",0,30));
                tnode.link(datasea.cl("Fall", "DN", "",-10,10));
                tnode.link(datasea.cl("Winter", "DN", "",10,10));
                tnode.link(datasea.cl("Spring", "DN", "",-10,10));
                tnode.link(datasea.cl("Summer", "DN", "",10,10));
// Cost node
        tnode = DM.link(datasea.cl("Cost", "AN", "Stats",20,30));
                tnode.link(datasea.cl("<10", "DN", "",-20,10));
                tnode.link(datasea.cl("<1000", "DN", "",-10,10));
                tnode.link(datasea.cl("<10000", "DN", "",0,10));
                tnode.link(datasea.cl(">10000", "DN", "",10,10));

tnode = datasea.find_node_named("Male");
for (i=0; i<5; i++)
        create_DM_DN("Male", "Sports", "Summer", "<100");
create_DM_DN("Male", "Sports", "Spring", "<1000");
create_DM_DN("Male", "Appliances", "Fall", "<10000");
create_DM_DN("Male", "Sports", "Summer", "<1000");

for (i=0; i<5; i++)
        create_DM_DN("Female", "Clothes", "Fall", "<1000");
create_DM_DN("Female", "Clothes", "Summer", "<100");
create_DM_DN("Female", "Appliances", "Summer", "<1000");
create_DM_DN("Female", "Appliances", "Winter", "<10000");
*****************************************************/

}// end popDM



/**
 **  create_DM_DN
 **
*******************************************************
public void create_DM_DN(String gender, String dept, String season, String cost)
{ // skip
        Node gender_node, dn;

gender_node = datasea.find_node_named(gender);
if (gender_node != null) {
        dn = gender_node.link(datasea.cl(""+global_counter++,"DN"));
        dn.link(dept); dn.link(season); dn.link(cost);
        }

} // end create_DM_DN

**********************************************/



/**
 **  con   Populate Econ
 **
*/
        public void popEcon () { // skip
        Node Econ_node, t_node;

/**********************************************
GUI.P(0,"con", "Economist Run.");
        Econ_node =
(DataSea.Root.link(datasea.cl("ECON","AN","Text",0,0)).link(datasea.cl("Economis
t This Week", "AN", "Text",0,10)));
        t_node = Econ_node.link(datasea.cl("ENTER NATO","AN","Text",0,20));
        (t_node.link(datasea.cl("NATO troops occupied KOSOVO after )"
                +"Serb troops had agreed to withdraw, but found that a contingent
of "
```

```
                +"Russian soldiers had reached the airport
first.","Text","Text",0,30)).link(datasea.cl("NATO","AN"));
        (t_node=t_node.link(datasea.cl("Despite some clashes between KLA fighters
and Serbs, ))"
                +"and between NATO soldiers and Serbs, NATO's takeover was
largely peaceful.","Text","Text",120,30)).link(datasea.cl("KLA","AN")));
        t_node.link(datasea.find_node_named("NATO"));

****************************************/

}// end popEcon


/**
 **  popURLs
 **
*/
public void popURLs () { // "show URLs"
Node URL0, URL1, URL2, URL3;

URL0 = create_node("URL:MSU-Bozeman Welcome", "URL");
URL0.Data[0] = "";
URL0.Data[1] = " Welcome to Montana State University at Bozeman!";
URL0.Data[2] = " ";
URL0.Data[3] = "Below you will find topics of interest.";
URL0.Data[4] = "";
URL0.Data[5] = "For enrollment information for Spring 2001, wait here.";
URL0.Data[6] = "  ";
URL0.Data[7] = "- William Blake Archive ";
URL0.Data[8] = "- Contact Student Services";
URL0.Data[9] = "- Search Faculty/Staff Directory";

URL1 = create_node("URL:Netscape:Directories", "URL");
URL1.Data[0] = "The area code for all Montana locations is 406. ";
URL1.Data[1] = "The MSU-Bozeman campus operator can be reached at ";
URL1.Data[2] = "                      (406)994-0211.  ";
URL1.Data[3] = " ";
URL1.Data[4] = "Montana State University ";
URL1.Data[5] = " ";
URL1.Data[6] = "Montana State University-Bozeman ";
URL1.Data[7] = "- Faculty/Staff Directory  ";
URL1.Data[8] = "- Student Directory  ";
URL1.Data[9] = "- Department Mailing Addresses  ";

URL2 = create_node("URL:MSU-Bozeman Faculty/Staff Directory ", "URL");
URL2.link(datasea.find_node_named("Dir"));
URL2.Data[0] = "";
URL2.Data[1] = " The faculty/staff directory is arranged alphabetically by last
name. ";
URL2.Data[2] = " ";
```

```
URL2.Data[3] = "A B C D E F G H I J K L M N O P Q R S T U V W X Y Z ";
URL2.Data[4] = "Choose a letter and then utilize the search capability of your
";
URL2.Data[5] = "browser to locate specific individuals within the directory ";
URL2.Data[6] = "section.  ";
URL2.Data[7] = " ";
URL2.Data[8] = "Please send any directory additions or corrections to Barb
Asper.  ";
URL2.Data[9] = "This directory was last updated July 27, 1999.  ";

URL3 = create_node("URL:MSU-Bozeman Faculty/Staff Directory (M)", "URL");
URL3.link(datasea.find_node_named("Dir"));
URL3.link(datasea.find_node_named("address"));
URL3.link(datasea.find_node_named("phone"));
URL3.link(datasea.find_node_named("email"));
URL3.Data[0] = "MSU-Bozeman Faculty/Staff Directory";
URL3.Data[1] = "MILLER, JOHN    278-7707";
URL3.Data[2] = "   RES ASSOC, AES-WEST TRI AG RES";
URL3.Data[3] = "   jpm@nervana.montana.edu";
URL3.Data[4] = "MILLER, JOHN    994-7332";
URL3.Data[5] = "   DIRECTOR-PHD, COMPUTATIONL BIOLOGY, AJ 29";
URL3.Data[6] = "   jpm@nervana.montana.edu";
URL3.Data[7] = "MILLER, KATHERINE   994-5067";
URL3.Data[8] = "   GRADUATE STUDENT, ENTOMOLOGY DEPARTMENT, LJ 415";
URL3.Data[9] = "   klmiller@montana.edu";


URL0.link(datasea.find_node_named("Web"));
URL1.link(URL0);
URL2.link(URL1);
URL3.link(URL2);
URL3.link(triplet("Miller","(406)994-7332","phone"));
triplet("Miller","jpm@nervana.montana.edu", "email");
triplet("Miller","123 Donnegal Dr, Bozeman, MT","address");
(datasea.find_node_named("Miller")).link(datasea.find_node_named("name"));

URL2 = create_node("URL:The William Blake Archive", "URL");
URL2.Data[0] = "";
URL2.Data[1] = " Known Hazards and Most Favorable Conditions of the Archive ";
URL2.Data[2] = " ";
URL2.Data[3] = " Graphical Help Screens (click icon for full-sized image):";
URL2.Data[4] = "";
URL2.Data[5] = "Help Table of Contents";
URL2.Data[6] = "- The Archive and the Web.  ";
URL2.Data[7] = "- Basic DynaWeb Navigation ";
URL2.Data[8] = "- - Table of Contents";
URL2.Data[9] = "- - - Collection Index";

URL2.link(URL0);
(create_node("Blake", "DN")).link(datasea.cl("name", "AN"));
```

```
URL2.link(datasea.find_node_named("Blake"));

GUI.P(0,"popURLs", "show web");
return;

} // end popURLs

/**
 **  popt   Populate time samples, 1,2,3 etc linked to dates, themselves linked
to TimeLine
 **
*/
public void popt () {
        int x, y, i, size_X, size_Y;

        GUI.P(0,"popt","time nodes Starting.");

        popTL();

        datasea.gui.tl.create_TS(create_node("day1", "Event"), null, "Jan 1
1999");
        datasea.gui.tl.create_TS(create_node("day2", "Event"), null, "Feb 2
1999");
        datasea.gui.tl.create_TS(create_node("day3", "Event"), null, "Mar 3
1999");
        datasea.gui.tl.create_TS(create_node("day4", "Event"), null, "Apr 4
1999");
        datasea.gui.tl.create_TS(create_node("day5", "Event"), null, "May 5
1999");
        datasea.gui.tl.create_TS(create_node("day6", "Event"), null, "Jun 6
1999");
        datasea.gui.tl.create_TS(create_node("day7", "Event"), null, "Jul 7
2000");
        datasea.gui.tl.create_TS(create_node("day8", "Event"), null, "Aug 8
2000");
        datasea.gui.tl.create_TS(create_node("day9", "Event"), null, "Sep 9
2000");
        datasea.gui.tl.create_TS(create_node("day10", "Event"), null, "Oct 10
2000");
        datasea.gui.tl.create_TS(create_node("day11", "Event"), null, "Nov 11
2000");
        datasea.gui.tl.create_TS(create_node("day12", "Event"), null, "Dec 12
2000");

        GUI.P(0,"popt","time nodes Done.");

} // end popt

/**
 **  popTL   Populate TimeLine
 **
*/
public void popTL () { // "show TL"
        int x, y, i, size_Y, size_X;
        int delta_x = 2; // offset from today_node, next_week, etc. to TL

        GUI.P(0,"L","TimeLine Run.");
// For demostration purposes,
// just code 'today', 'next-week', etc, as hard-coded dates

        //y=10;
        size_X=1;
        size_Y=(int)(TimeLine.size_Y * 0.3);
        //x=(int)(TimeLine.size_Y * 0.01);

        yesterday_node = new Node("Yesterday", "Event", "Yesterday");
        Node tn = datasea.gui.tl.create_TS(yesterday_node,null,"May 21 2000");
        yesterday_node.Y = tn.Y - 15;
        yesterday_node.X = TimeLine.X - delta_x;
        yesterday_node.size_Y = 3;
        yesterday_node.size_X = 1;

        today_node = new Node("Today", "Event", "Today");
        tn = datasea.gui.tl.create_TS(today_node,null,"May 22 2000");
        today_node.Y = tn.Y;
        today_node.X = TimeLine.X - delta_x;
        today_node.size_Y = 3;
        today_node.size_X = 1;

        tomorrow_node = new Node("Tomorrow", "Event", "Tomorrow");
        tn = datasea.gui.tl.create_TS(tomorrow_node,null,"May 23 2000");
        tomorrow_node.Y = tn.Y + 15;
        tomorrow_node.X = TimeLine.X - delta_x;
        tomorrow_node.size_Y = 3;
        tomorrow_node.size_X = 1;

        last_week_node = new Node("LastWeek", "Event", "last_week");

        datasea.gui.tl.create_TS(last_week_node,null,"May 2 2000");
        datasea.gui.tl.create_TS(last_week_node,null,"May 3 2000");
        datasea.gui.tl.create_TS(last_week_node,null,"May 4 2000");
        datasea.gui.tl.create_TS(last_week_node,null,"May 5 2000");
        tn = datasea.gui.tl.create_TS(last_week_node,null,"Mar 6 2000");
        last_week_node.Y = tn.Y-60;
        last_week_node.X = TimeLine.X - 2*delta_x;
        last_week_node.size_Y = 30;
        last_week_node.size_X = 1;
        datasea.gui.tl.create_TS(last_week_node,null,"May 7 2000");
        datasea.gui.tl.create_TS(last_week_node,null,"May 8 2000");
        datasea.gui.tl.create_TS(last_week_node,null,"May 9 2000");
```

```
            this_week_node = new Node("ThisWeek", "Event", "this_week");

            datasea.gui.tl.create_TS(this_week_node,null,"May 22 2000");
            datasea.gui.tl.create_TS(this_week_node,null,"May 23 2000");
            datasea.gui.tl.create_TS(this_week_node,null,"May 24 2000");
            datasea.gui.tl.create_TS(this_week_node,null,"May 25 2000");
            tn = datasea.gui.tl.create_TS(this_week_node,null,"May 14 2000");
            this_week_node.Y = tn.Y;
            this_week_node.X = TimeLine.X - 2*delta_x;
            this_week_node.size_Y = 30;
            this_week_node.size_X = 1;
            datasea.gui.tl.create_TS(this_week_node,null,"May 26 2000");
            datasea.gui.tl.create_TS(this_week_node,null,"May 27 2000");
            datasea.gui.tl.create_TS(this_week_node,null,"May 28 2000");


            next_week_node = new Node("NextWeek", "Event", "next_week");

            datasea.gui.tl.create_TS(next_week_node,null,"May 29 2000");
            datasea.gui.tl.create_TS(next_week_node,null,"May 30 2000");
            datasea.gui.tl.create_TS(next_week_node,null,"May 31 2000");
            datasea.gui.tl.create_TS(next_week_node,null,"Jun 1 2000");
            tn = datasea.gui.tl.create_TS(next_week_node,null,"Mar 22 2000");
            next_week_node.Y = tn.Y + 60;
            next_week_node.X = TimeLine.X - 2*delta_x;
            next_week_node.size_Y = 30;
            next_week_node.size_X = 1;
            datasea.gui.tl.create_TS(next_week_node,null,"Jun 2 2000");
            datasea.gui.tl.create_TS(next_week_node,null,"Jun 3 2000");
            datasea.gui.tl.create_TS(next_week_node,null,"Jun 4 2000");

            datasea.gui.tl.create_date_node("Feb 1 2000");
            datasea.gui.tl.create_date_node("Apr 1 2000");
            datasea.gui.tl.create_date_node("May 1 2000");
            datasea.gui.tl.create_date_node("Jun 1 2000");
            datasea.gui.tl.create_date_node("Jul 1 2000");
            datasea.gui.tl.create_date_node("Aug 1 2000");
            datasea.gui.tl.create_date_node("Sep 1 2000");
            datasea.gui.tl.create_date_node("Oct 1 2000");
            datasea.gui.tl.create_date_node("Nov 1 2000");
            datasea.gui.tl.create_date_node("Dec 1 2000");

    GUI.P(0, "popTL", "show TL");
    return;
    } // end TL


    /**
```

```
 **  popcalendar
 **
*/
public void popcalendar () {
int i, size;
Node child;

GUI.input.string_input("input Open House Sue&Fred Jan 1 2000");
GUI.input.string_input("input Bay Photo pickup Jan 3 2000");
GUI.input.string_input("input Bay Photo pickup Feb 3 2000");
GUI.input.string_input("input Custom Process Photo pickup Mar 30 2000");
GUI.input.string_input("input Bay Photo pickup Apr 23 2000");
GUI.input.string_input("input Darkroom 1 3-5pm Jan 15 2000");
GUI.input.string_input("input ASUC Photo exhibit reception  6-7pm Mar 24 2000");
GUI.input.string_input("input SOMArts exhibit reception  5-7pm Apr 6 2000");
GUI.input.string_input("input Tara & Gary BA 5:45pm Jan 7 2000");
GUI.input.string_input("input Rozalina Party 8pm Jan 14 2000");
GUI.input.string_input("input Lauren 2pm photo shoot Jan 20 2000");
GUI.input.string_input("input Lauren 1:30pm photo shoot Feb 8 2000");
GUI.input.string_input("input Dinner with Erin 7pm Mar 2 2000");
GUI.input.string_input("input Cal' Bach Society concert 9pm St Marks Mar 4
2000");
GUI.input.string_input("input Watt's master class SF Conservatory 11:30am Mar 6
2000");
GUI.input.string_input("input SFEMS Concert 8pm Mar 11 2000");
GUI.input.string_input("input SF FA Museum Floral exhibit Mar 14 2000");
GUI.input.string_input("input Sherman-Clay piano sale, UCB 1pm Mar 16 2000");
GUI.input.string_input("input Dinner with Erin 7pm Apr 9 2000");

} // end popcalendar


/**
 **  calendar
 **
*/
       public void calendar () {
        int i,j,day=0;
        Node cal_node, tnode;

cal_node = new Node("cal", "DN", "Calendar", 0, 0, 200,150);
cal_node.set_mag(0.5);
//cal_node.TinyScale = 0.1;
cal_node.link(TimeLine);
datasea.set_child_position(TimeLine, cal_node, 0, -300);

for (i=1; i<6; i++)
       for (j=1; j<8; j++) {
              if (++day > 31)
                     break;
```

```
                tnode = new Node(""+day, "DN", "Day", 0, 0, 30,20);
                tnode.link(cal_node);
                tnode.set_mag(0.9);
                datasea.set_child_position(cal_node, tnode, ((double)(j))/8.0,
0.1+((double)(5-i))/5.0);
        }

GUI.P(0, "popcalendar", "show cal");
} // end calendar



/*
 **
 */
public void add_directory_entry (String input_string) {
Node name=null, address=null, phone=null, other=null, email=null;
Node Directory, name_an_node, address_an_node, phone_an_node, email_an_node;
int num_words, i;
String words[];

Node DirCNode = create_node("DirCN","CN");
Node MailDirCNode = create_node("MailDirCN","CN");

        StringTokenizer t = new StringTokenizer(input_string, " ,.<>\"\t\r\n" );
        num_words = t.countTokens();
        words = new String[num_words];
        for(i = 0; i < num_words; i++)
                words[i] = t.nextToken();

Directory = datasea.find_node_named("Dir");
if (Directory==null) {
        Directory = new Node("Dir","AN","");
        DataSea.Root.link(Directory, "polarized");
        GUI.P(1,"add_directory_entry", " Creating Directory, linking it to
DataSea.Root ");
        }

name_an_node = datasea.find_node_named("name");
if (name_an_node==null) {
        name_an_node = new Node("name","AN","");
        }
Directory.link(name_an_node, DirCNode, "polarized");

address_an_node = datasea.find_node_named("address");
if (address_an_node==null) {
        address_an_node = new Node("address","AN","");
        }
Directory.link(address_an_node, DirCNode, "polarized");
```

```
phone_an_node = datasea.find_node_named("phone");
if (phone_an_node==null) {
       phone_an_node = new Node("phone","AN","");
       //Node telephone_an_node = new Node("telephone","AN","");
       //telephone_an_node.link(phone_an_node);
       }
Directory.link(phone_an_node, DirCNode, "polarized");

email_an_node = datasea.find_node_named("email");
if (email_an_node==null) {
       email_an_node = new Node("email","AN","");
       }
//Directory.link(email_an_node, MailDirCNode, "polarized");

if (num_words >= 1) {
       name = datasea.find_node_named(words[0]);
       if (name==null)
              name=new Node(words[0], "DN", "name");
       name_an_node.link(name, "polarized");
       //name_an_node.link(name, DirCNode, "polarized");
}
if (num_words >= 2) {
       phone = datasea.find_node_named(words[1]);
       if (phone==null)
              phone=new Node(words[1], "DN", "phone");
       phone_an_node.link(phone, "polarized");
       //phone_an_node.link(phone, DirCNode, "polarized");
}
if (num_words >= 3) {
       address = datasea.find_node_named(words[2]);
       if (address==null)
              address=new Node(words[2], "DN", "address");
       address_an_node.link(address, "polarized");
       //address_an_node.link(address, DirCNode, "polarized");
       }
if (num_words >= 4) {
       email = datasea.find_node_named(words[3]);
       if (email==null)
              email=new Node(words[3], "DN", "email");
       email_an_node.link(email, MailDirCNode, "polarized");
}

       name.link(phone, DirCNode, "unpolarized");
       name.link(address, DirCNode, "unpolarized");
       name.link(email, DirCNode, "unpolarized");

       datasea.gui.show_node_once(name);
       datasea.gui.show_node_once(phone);
       datasea.gui.show_node_once(address);
       datasea.gui.show_node_once(email);
```

```
//datasea.set_CSs_of_DirCNode(DirCNode, 10);
//datasea.set_CSs_of_DirCNode(MailDirCNode, 30);


} // end add_directory_entry



 /**
 **  DIRECTORY
 **
*/
        public void popd () { // "show Dir"
        Node tn;

add_directory_entry("Abe 111-1111 1111_Abner abe@aol.com");
add_directory_entry("Bob 222-2222 2222_Broadway bob@aol.com");
add_directory_entry("Carl 333-3333 1234_Claremont carl@aol.com");
add_directory_entry("Dave 444-4444 1234_Denison dave@aol.com");
add_directory_entry("Evan 555-5555 1234_Edgar evan@aol.com");
add_directory_entry("Frank 666-6666 1234_Fairway frank@aol.com");
add_directory_entry("Rocky 888-8888 1234_Campus rocky@tallis.com");



datasea.needdistUpdate = true;
GUI.P(0,"popd", "show Dir   or   Bob");
return;
} // end  popd




/**
 **  popportal
 **
**********************************************************************
        public void popportal () { // "show Yahoo"
        Node n1, n2, n3, n4, n5, n6, n7, n8;
System.err.println("Populate.portal() begun");

Node Yahoo = create_node("Yahoo","AN");
ANpair("Yahoo","Animal");
ANpair("Animal","FoodAnimals");
ANpair("FoodAnimals","Poultry");
ANpair("FoodAnimals","Cattle");
ANpair("Animal","Reproduction");
ANpair("Reproduction","Egg");
ANpair("Egg","EggTempera");
ANpair("Egg","Reproduction");
ANpair("Egg","Poultry");
ANpair("Animal","Reproduction");
ANpair("EggTempera","ETRecipes");
ANpair("EggTempera","Longevity");
ANpair("EggTempera","Longevity");
ANpair("EggTempera","OldMaster_Techniques");
ANpair("EggTempera","Techniques");
ANpair("EggTempera","ArtMedia");
ANpair("Animal","Morphology");
ANpair("Morphology","Skeletal");
ANpair("Morphology","Circulation");
ANpair("Morphology","Skin");
ANpair("Morphology","Morphology");
ANpair("Yahoo","Art");
ANpair("Art","ArtMaterials");
ANpair("Art","ArtHistory");
ANpair("Art","Artists");
ANpair("Art","Painting");
ANpair("Animal","Environment");
ANpair("ArtMaterials","ArtMedia");
ANpair("Painting", "Frescoe");
ANpair("Painting", "OilPaintings");
ANpair("Painting", "WaterColors");
ANpair("Frescoe","EggTempera");

ANDNpair("Poultry","EggProduction");
ANDNpair("Egg","EggProduction");

ANDNpair("ArtMedia","Artmedia&Frescoe");
ANDNpair("Frescoe","Artmedia&Frescoe");

ANDNpair("EggTempera","ET&Frescoe");
ANDNpair("Frescoe","ET&Frescoe");

ANDNpair("OilPaintings","OldMasterPaintings");
ANDNpair("Frescoe","OldMasterPaintings");

ANDNpair("Egg","Egg&Skin");
ANDNpair("Skin","Egg&Skin");

ANDNpair("Morphology","Skin&Morph");
ANDNpair("Skin","Skin&Morph");

ANDNpair("Reproduction","EggIncubation");
ANDNpair("Egg","EggIncubation");
ANDNpair("Egg","EggNutrition");
ANDNpair("FoodAnimals","EggNutrition");
ANDNpair("Poultry","EggNutrition");
ANDNpair("Reproduction","LiveBirth");
ANDNpair("Animal","Animal&Morph");
ANDNpair("Morphology","Animal&Morph");

/////////////////////////  search result node linked to 100 URLS
/////////////////////////  group of 100 URLs
/////////////////////////  ANs distal
/////////////////////////
/////////////////////////  user selects ANs, samples URLS, adds more search terms
+/-
/////////////////////////

GUI.P(0, "popportal", "show Yahoo");

popmany();

System.err.println("Populate.portal() done.");
System.err.println("Populate.portal() Yahoo ="+Yahoo);
} // end popportal



/**
 **  popmany   like popportal, but with more links to each DN
 **

        public void popmany () { // "show Yahoo"
        Node n1, n2, n3, n4, n5, n6, n7, n8;
// Node Art, ArtMaterials, Frescoe, EggTempera OldMaster_Techniques;
// Node Egg, Animal, Reproduction, Morphology, Circulation, Skeletal, Skin;

Node Yahoo = create_node("xYahoo","AN");
Node Egg = create_node("xEgg", "AN");
Node Animal = create_node("xAnimal", "AN");
Node Reproduction = create_node("xReproduction", "AN");
Node Morphology = create_node("xMorphology", "AN");
Node Circulation = create_node("xCirculation", "AN");
Node Skeletal = create_node("xSkeletal", "AN");
Node Skin = create_node("xSkin", "AN");
Node Art = create_node("xArt", "AN");
Node ArtMaterials = create_node("xArtMaterials", "AN");
Node Frescoe = create_node("xFrescoe", "AN");
Node EggTempera = create_node("xEggTempera", "AN");
Node OldMaster_Techniques = create_node("xOldMaster_Techniques", "AN");
Node ArtMedia = create_node("xArtMedia", "AN");
Node Recipes = create_node("xRecipes", "AN");
Node Longevity = create_node("xLongevity", "AN");
Node Techniques = create_node("xTechniques", "AN");
Node Environment = create_node("xEnvironment", "AN");
Node LiveBirth = create_node("xLiveBirth", "AN");
Node Poultry = create_node("xPoultry", "AN");
Node ArtHistory = create_node("xArtHistory", "AN");
Node Artists = create_node("xArtists", "AN");

Yahoo.link(Animal, "polarized");
Yahoo.link(Art, "polarized");
Egg.link(EggTempera, "polarized");
Egg.link(Reproduction, "polarized");
Egg.link(Poultry, "polarized");
Animal.link(Reproduction, "polarized");
Animal.link(Environment, "polarized");
Animal.link(Morphology, "polarized");
EggTempera.link(Recipes, "polarized");
EggTempera.link(Longevity, "polarized");
EggTempera.link(OldMaster_Techniques, "polarized");
EggTempera.link(Techniques, "polarized");
EggTempera.link(ArtMedia, "polarized");
Morphology.link(Skeletal, "polarized");
Morphology.link(Circulation, "polarized");
Morphology.link(Skin, "polarized");
Reproduction.link(LiveBirth, "polarized");
Art.link(ArtMaterials, "polarized");
Art.link(ArtHistory, "polarized");
Art.link(Artists, "polarized");
ArtMaterials.link(ArtMedia, "polarized");
Frescoe.link(EggTempera, "polarized");


////////////////////////  search result node linked to 100 URLS
////////////////////////  group of 100 URLs
////////////////////////  ANs distal
////////////////////////
////////////////////////  user selects ANs, samples URLS, adds more search terms
+/-
////////////////////////

int counter=0;
Node dn = create_node("xArtmedia&Frescoe", "DN");
ArtMedia.link(dn);
Frescoe.link(dn);
Art.link(dn);
EggTempera.link(dn);

dn = create_node("xET&Frescoe", "DN");
ArtMedia.link(dn, "polarized");
Frescoe.link(dn, "polarized");
Art.link(dn, "polarized");
EggTempera.link(dn, "polarized");

dn = create_node("xEgg&Skin", "DN");
```

```
Egg.link(dn, "polarized");
Skin.link(dn, "polarized");
LiveBirth.link(dn, "polarized");
Reproduction.link(dn, "polarized");


dn = create_node("xSkin&Morph", "DN");
Morphology.link(dn, "polarized");
Skin.link(dn, "polarized");


dn = create_node("xAnimal&Morph", "DN");
Animal.link(dn, "polarized");
Morphology.link(dn, "polarized");
Environment.link(dn, "polarized");
Reproduction.link(dn, "polarized");

GUI.P(0, "popmany", "show xYahoo");

} // end popmany
************************************************/




/**
 **  popframe Christmas-tree frame for other data to attach to
 **
*/
        public void popframe () { // "show n1"
        Node n1, n2, n3, n4, n5, n6, n7, n8;
//String type = "BB";
String type = "DN";

n1 = create_node("n1", type);
n1.X=40;
n1.Y=50;
datasea.Root.link(n1);

///////////////////  VERTICAL PIECES
n2 = create_node("n2", type);
n2.link(n1);
        n2.X=0;
        n2.Y = 75;
n3 = create_node("n3", type);
n3.link(n2);
        n3.X=0;
        n3.Y = 75;
///////////////////
///////////////////  HORIZONTAL PIECES
```

```
n4 = create_node("n4", type);
n4.link(n3);
        n4.X=50;
        n4.Y = -25;
n5 = create_node("n5", type);
n5.link(n3);
        n5.X=-50;
        n5.Y = -25;
n6 = create_node("n6", type);
n6.link(n2);
        n6.X=50;
        n6.Y = -25;
n7 = create_node("n7", type);
n7.link(n2);
        n7.X=-50;
        n7.Y = -25;

GUI.P(0, "popframe", "show n1");
return;
} // end popframe



/**
 **  popp
 **
 **          creates mail,url and note, links them to lastweek,yesterday and
today
*/
public void popp () {
int i, size;
Node child;

// Mary's EMAIL about printer
Node mail = email("Mary EMAIL I really like HP Printer 5MP");
datasea.gui.tl.create_TS(mail,null,"Jul 3 2000");

// Bob's NOTE about printer 'A' in Joe Baker's Office
Node n = GUI.input.string_input("input Bob said it would be $300 to fix Joe
Baker Printer named 'A'");


// URL advertisement for HP printer
Node url_node = create_node("Ad for HP Printer 5MP", "URL");
url_node.Desc="HP Advertisement";
url_node.link(datasea.find_node_named("Printer"));
url_node.link(datasea.find_node_named("5MP"));
url_node.link(datasea.find_node_named("HP"));
// url_node.link(datasea.find_node_named("Yesterday"));
datasea.gui.tl.create_TS(url_node,null,"Jul 9 2000");
```

```
//poppp();

return;
} // end popp


/**
 **  poppp
 **
*/
public void poppp () {
int i, size;
Node child;

// Blah EMAIL
email("Bob EMAIL This is an interesting day, my cat is happy");
email("Bob EMAIL Yesterday is an interesting day, my dog is happy");
email("Bob EMAIL Tomorrow is an interesting day, my parrot is happy");
email("Bob EMAIL LastWeek was an interesting week, my cougar is happy");

// Blah URL
Node url_node = create_node("kjkajdf kjasdfkjasdkf ", "URL");
url_node.Desc="URL Blah blah";
url_node.link(datasea.find_node_named("blah"));

// Blah URL
url_node = create_node("kjksa asdkjasf kasf", "URL");
url_node.Desc="URL akjasdkf kasdf lasfd something ";
url_node.link(datasea.find_node_named("LastWeek"));

// Blah URL
url_node = create_node("ak jkska l aslas dfs ", "URL");
url_node.Desc="URL xxkk asdkf asdjfas";
url_node.link(datasea.find_node_named("xxkk "));
url_node.link(datasea.find_node_named("Tomorrow"));

// Blah Notes ...
GUI.input.string_input("input Abe believes in flying saucers. Jan 5 1999");
GUI.input.string_input("input Abe thinks of nothing . Feb 10 1999");
GUI.input.string_input("input Bob says everything. Mar 20 1999");
GUI.input.string_input("input Bob drives a car. Jan 25 2000");
GUI.input.string_input("input Carl does nothing. Mar 30 2000");

return;
} // end poppp


/**
 **  email
 **
*/
public Node email (String input_string) {
int i, size;
Node child;
String individual_word;

Node email_node = create_node("email");
Node mail = create_node(input_string, "DN");

String useful_string = datasea.gui.input.discard_words(input_string);
StringTokenizer t = new StringTokenizer(useful_string, " ");
int num_words = t.countTokens();

for(i = 0; i < num_words; i++) {
       individual_word = t.nextToken();
       mail.link(create_node(individual_word, "DN"));
       }

return(mail);
} // end email

/**
 **  popnet
 **
*/
public void popnet () {
int i, size;
Node child;

pull_in_URLs("http://www.paintedeggs.com/","eggs2.html", 3, (Node)null);
pull_in_URLs("http://www.alvr.com/","eggs/emainpage.html", 3, (Node)null);
/**************************************
pull_in_URLs("http://www.eggtempera.com/","faq.html", 1, (Node)null);
pull_in_URLs("http://netvet.wustl.edu/", "birds.htm", 1, (Node)null);
pull_in_URLs("http://www.berkeley.edu/", "index.html", 1, (Node)null);
**************************************/
/**************************************
pull_in_URLs("http://www.eggtempera.com/","welcome.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","stpoptions.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","aboutsociety.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","aboutet.html" , 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","history.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","publications.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","instructors.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","materials.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","guestbook.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","links.html" , 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","intro.html", 1, (Node)null);
**************************************/
```

```
} // end popnet

/**
 **  popweb   sample of web history
 **
*/
        public void popweb () { // "show WebHistory"
        Node l, m;
        int counter = 0;
        int counter_delta=2;

l = create_node("WebHistory", "DN");
l.X=20;
l.Y=counter-=counter_delta;
datasea.Root.link(l);
l.link(datasea.find_node_named("Web"));

m = create_node("Search:EggTempera", "DN");
m.X=20;
m.Y=counter-=counter_delta;
l.link(m);

Node n1 = create_node("http://www.eggtempera.com/welcome.html", "URL");
Node n2 = create_node("http://www.eggtempera.com/stpoptions.html", "URL");
Node n3 = create_node("http://www.eggtempera.com/aboutsociety.html", "URL");
Node n4 = create_node("http://www.eggtempera.com/aboutet.html", "URL");
Node n5 = create_node("http://www.eggtempera.com/history.html", "URL");
Node n6 = create_node("http://www.eggtempera.com/publications.html", "URL");
Node n7 = create_node("http://www.eggtempera.com/instructors.html", "URL");
Node n8 = create_node("http://www.eggtempera.com/faq.html", "URL");
Node n9 = create_node("http://www.eggtempera.com/materials.html", "URL");
Node n10 = create_node("http://www.eggtempera.com/guestbook.html", "URL");
Node n11 = create_node("http://www.eggtempera.com/links.html", "URL");
Node n12 = create_node("http://www.eggtempera.com/intro.html", "URL");

m.link(n1);
n1.link(n2);
n2.link(n3);
n3.link(n4);
n4.link(n5);
n5.link(n6);
n6.link(n7);
n7.link(n8);
n8.link(n9);
n9.link(n10);
n10.link(n11);
n11.link(n12);
```

```
n5.link(datasea.find_node_named("x5"));
n7.link(datasea.find_node_named("x7"));

n1.X=0; n1.Y=counter-=counter_delta;
n2.X=0; n2.Y=counter-=counter_delta;
n3.X=0; n3.Y=counter-=counter_delta;
n4.X=0; n4.Y=counter-=counter_delta;
n5.X=0; n5.Y=counter-=counter_delta;
n6.X=0; n6.Y=counter-=counter_delta;
n7.X=0; n7.Y=counter-=counter_delta;
n8.X=0; n8.Y=counter-=counter_delta;
n9.X=0; n9.Y=counter-=counter_delta;
n10.X=0; n10.Y=counter-=counter_delta;
n11.X=0; n11.Y=counter-=counter_delta;
n12.X=0; n12.Y=counter-=counter_delta;

Node cn = create_node("WebSession.1", "CN");
cn.link(n1);
cn.link(n2);
cn.link(n3);
cn.link(n4);
cn.link(n5);
cn.link(n6);
cn.link(n7);
cn.link(n8);
cn.link(n9);
cn.link(n10);
cn.link(n11);
cn.link(n12);
//cn.link(datasea.find_node_named("today"));

GUI.P(0,"popweb", "show WebHistory");

return;
} // end popweb



/**
 **  poprev
 **
*/
public void poprev () {
int i, size;
Node child;

Node ruth = create_node("Ruth");
Node A1 = create_node("A1");
Node A2 = create_node("A2");
Node A3 = create_node("A3");
Node A4 = create_node("A4");
Node target = create_node("target");
Node B1 = create_node("B1");

ruth.link(A1);
A1.link(A2);
A2.link(A3);
A3.link(A4);
A4.link(target);
ruth.link(B1);
B1.link(A4);

//  ruth - A1 - A2 - A3 - A4 - target
//     \--B1 ------------/
// set_Tdist_start(ruth, target)   should order them thus:
//
// 1  -  2  -  3  -  4  -  5  -  6
//     \  2  - - - - - - -/

} // end poprev

/**
 **  popegg
 **
*/
public void popegg () {
int i, size;
Node child;


pull_in_URLs("http://www.eggtempera.com/","welcome.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","stpoptions.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","aboutsociety.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","aboutet.html" , 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","history.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","publications.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","instructors.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","faq.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","materials.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","guestbook.html", 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","links.html" , 1, (Node)null);
pull_in_URLs("http://www.eggtempera.com/","intro.html", 1, (Node)null);

} // end popegg



/**
 **  popBB   BackBone network
 **
```

```
*/
        public void popBB () { // "show Lessons"
        Node l, m, prior, x,y,z,book_node;
        int counter = -30;
        int counter_delta = 3;


Node CN1 = new Node("BackBoneCN", "CN");

l = create_node("Lessons", "DN");
l.Y=counter+=10;
l.X=20;
datasea.Root.link(l);


l.link(make_BB("start","Lesson1"));
        prior = lastBBnode;
        lastBBnode.Y = counter+=counter_delta;
        lastBBnode.X=20;
make_BB("add","Chapter_1");
        prior = lastBBnode;
        CN1.link(prior);
        lastBBnode.Y = counter+=counter_delta;
        lastBBnode.X=20;
make_BB("add","Chapter_2");
        prior.link(book_node = create_node("Some Book1", "DN"));
        CN1.link(prior);
        x=create_node("x1","DN");
        y=create_node("y1","DN");
        z=create_node("z1","DN");
        book_node.link(x);
        x.link(y);
        y.link(z);
        prior = lastBBnode;
        lastBBnode.Y = counter+=counter_delta;
        lastBBnode.X=20;
make_BB("add","Chapter_3");
        prior.link(book_node = create_node("Some Book2", "DN"));
        CN1.link(prior);
        x=create_node("x2","DN");
        y=create_node("y","DN");
        z=create_node("z2","DN");
        book_node.link(x);
        x.link(y);
        y.link(z);
        prior = lastBBnode;
        lastBBnode.Y = counter+=counter_delta;
        lastBBnode.X=20;
make_BB("add","Chapter_4");
        prior.link(book_node = create_node("Some Book3", "DN"));
```

```
          CN1.link(prior);
          x=create_node("x3","DN");
          y=create_node("y3","DN");
          z=create_node("z3","DN");
          book_node.link(x);
          x.link(y);
          y.link(z);
          prior = lastBBnode;
          lastBBnode.Y = counter+=counter_delta;
          lastBBnode.X=20;
  make_BB("add","Chapter_5");
          prior.link(book_node = create_node("Some Book4", "DN"));
          CN1.link(prior);
          x=create_node("x4","DN");
          y=create_node("y4","DN");
          z=create_node("z4","DN");
          book_node.link(x);
          x.link(y);
          y.link(z);
          prior = lastBBnode;
          lastBBnode.Y = counter+=counter_delta;
          lastBBnode.X=20;
  make_BB("add","Chapter_6");
          prior.link(book_node = create_node("Some Book5", "DN"));
          CN1.link(prior);
          x=create_node("x5","DN");
          y=create_node("y5","DN");
          z=create_node("z5","DN");
          book_node.link(x);
          x.link(y);
          y.link(z);
          prior = lastBBnode;
          lastBBnode.Y = counter+=counter_delta;
          lastBBnode.X=20;
  make_BB("add","Chapter_7");
          prior.link(book_node = create_node("Some Book6", "DN"));
          CN1.link(prior);
          x=create_node("x6","DN");
          y=create_node("y6","DN");
          z=create_node("z6","DN");
          book_node.link(x);
          x.link(y);
          y.link(z);
          prior = lastBBnode;
          lastBBnode.Y = counter+=counter_delta;
          lastBBnode.X=20;
  make_BB("add","Chapter_8");
          prior.link(book_node = create_node("Some Book7", "DN"));
          CN1.link(prior);
          x=create_node("x7","DN");
          y=create_node("y7","DN");
          z=create_node("z7","DN");
          book_node.link(x);
          x.link(y);
          y.link(z);
          prior = lastBBnode;
          lastBBnode.Y = counter+=counter_delta;
          lastBBnode.X=20;
  make_BB("add","Chapter_9");
          prior.link(book_node = create_node("Some Book8", "DN"));
          CN1.link(prior);
          x=create_node("x8","DN");
          y=create_node("y8","DN");
          z=create_node("z8","DN");
          book_node.link(x);
          x.link(y);
          y.link(z);
          lastBBnode.link(book_node = create_node("Some Book9", "DN"));
          lastBBnode.Y = counter+=counter_delta;
          lastBBnode.X=20;


  Node CN2 = new Node("BackBoneCN", "CN");
  l.link(make_BB("start","Lesson2"));
          prior = lastBBnode;
          lastBBnode.Y = counter+=counter_delta;
          lastBBnode.X=20;
  make_BB("add","xChapter_1");
          prior = lastBBnode;
          lastBBnode.Y = counter+=counter_delta;
          lastBBnode.X=20;
  make_BB("add","xChapter_2");
          prior.link(book_node = create_node_forced("Another Book1", "DN"));
          CN2.link(prior);
          x=create_node_forced("xx1","DN");
          y=create_node_forced("yy1","DN");
          z=create_node_forced("zz1","DN");
          book_node.link(x);
          x.link(y);
          y.link(z);
          prior = lastBBnode;
          lastBBnode.Y = counter+=counter_delta;
          lastBBnode.X=20;
  make_BB("add","xChapter_3");
          prior.link(book_node = create_node_forced("Another Book2", "DN"));
          CN2.link(prior);
          x=create_node_forced("xx2","DN");
          y=create_node_forced("yy","DN");
          z=create_node_forced("zz2","DN");
          book_node.link(x);
```

```
        x.link(y);
        y.link(z);
        prior = lastBBnode;
        lastBBnode.Y = counter+=counter_delta;
        lastBBnode.X=20;
make_BB("add","xChapter_4");
        prior.link(book_node = create_node_forced("Another Book3", "DN"));
        CN2.link(prior);
        x=create_node_forced("xx3","DN");
        y=create_node_forced("yy3","DN");
        z=create_node_forced("zz3","DN");
        book_node.link(x);
        x.link(y);
        y.link(z);
        prior = lastBBnode;
        lastBBnode.Y = counter+=counter_delta;
        lastBBnode.X=20;
make_BB("add","xChapter_5");
        prior.link(book_node = create_node_forced("Another Book4", "DN"));
        CN2.link(prior);
        x=create_node_forced("xx4","DN");
        y=create_node_forced("yy4","DN");
        z=create_node_forced("zz4","DN");
        book_node.link(x);
        x.link(y);
        y.link(z);
        prior = lastBBnode;
        lastBBnode.Y = counter+=counter_delta;
        lastBBnode.X=20;
make_BB("add","xChapter_6");
        prior.link(book_node = create_node_forced("Another Book5", "DN"));
        CN2.link(prior);
        x=create_node_forced("xx5","DN");
        y=create_node_forced("yy5","DN");
        z=create_node_forced("zz5","DN");
        book_node.link(x);
        x.link(y);
        y.link(z);
        prior = lastBBnode;
        lastBBnode.Y = counter+=counter_delta;
        lastBBnode.X=20;
make_BB("add","xChapter_7");
        prior.link(book_node = create_node_forced("Another Book6", "DN"));
        CN2.link(prior);
        x=create_node_forced("xx6","DN");
        y=create_node_forced("yy6","DN");
        z=create_node_forced("zz6","DN");
        book_node.link(x);
        x.link(y);
        y.link(z);
        prior = lastBBnode;
        lastBBnode.Y = counter+=counter_delta;
        lastBBnode.X=20;
make_BB("add","xChapter_8");
        prior.link(book_node = create_node_forced("Another Book7", "DN"));
        CN2.link(prior);
        x=create_node_forced("xx7","DN");
        y=create_node_forced("yy7","DN");
        z=create_node_forced("zz7","DN");
        book_node.link(x);
        x.link(y);
        y.link(z);
        prior = lastBBnode;
        lastBBnode.Y = counter+=counter_delta;
        lastBBnode.X=20;
make_BB("add","xChapter_9");
        prior.link(book_node = create_node_forced("Another Book8", "DN"));
        CN2.link(prior);
        x=create_node_forced("xx8","DN");
        y=create_node_forced("yy8","DN");
        z=create_node_forced("zz8","DN");
        book_node.link(x);
        x.link(y);
        y.link(z);
        lastBBnode.link(book_node = create_node_forced("Another Book9", "DN"));
        lastBBnode.Y = counter+=counter_delta;
        lastBBnode.X=20;

GUI.P(0, "popBB", "show Lessons");
} // end popBB


/**
 **  pops   SIMILAR NODES
 **
*/
        public void pops () { // "show egg-tempera"
        Node node_sim;
        Node et, gt, si, f, food, acrylics, oils;
        Node AP, PT, E, RP, RI;

GUI.P(0,"", " late similarities, e.g. Egg-Tempera ");
//      node_sim=create_node("sim", "AN", "");

        et=create_node("Egg-Tempera", "DN", "");
        gt=create_node("Glazing_Techniques", "DN", "");
        si=create_node("Secular_Images", "DN", "");
        f=create_node("Frescoes", "DN", "");
        food=create_node("food", "DN", "");
        acrylics=create_node("acrylics", "DN", "");
        oils=create_node("oils", "DN", "");

        AP=create_node("Art_Practice", "AN", "");
        PT=create_node("Painting_Techniques", "AN", "");
        E=create_node("Eggs", "AN", "");
        RP=create_node("Renaissance-Paintings", "AN", "");
        RI=create_node("Religious_Images", "AN", "");

        AP.link(et);
        PT.link(et);
        E.link(et);
        RP.link(et);
        RI.link(et);

        AP.link(gt);
        PT.link(gt);
        AP.link(f);
        PT.link(f);
        AP.link(acrylics);
        PT.link(acrylics);
        AP.link(oils);
        PT.link(oils);
        E.link(food);
        RP.link(si);
        RP.link(f);
        RI.link(f);

datasea.needdistUpdate = true;
GUI.P(0, "pops", "show egg-tempera");
return;
} // end  (SIMILAR NODES)


/**
 **  popm   MAIL
 **
*/
        public void popm () { // "show email"
         int i;
        Node email, read, unread;

DataSea.currentCNode = create_node("MailCN", "CN"); // Turn on auto-CNode links

email("Hi Abe, this is email written about dogs.");
email("Dear Mary, How's the project going?");
email("Bob: please turn in your budget for next year.");
email("Carl: don't forget to close the door.");

DataSea.currentCNode = null; // Turn off auto-CNode links
```

```
datasea.needdistUpdate = true;
GUI.P(0, "popm", "show email");
return;
} // end  (pop-email)


/**
 **  get_node_from_file_suffix
 **
*/
public Node get_node_from_file_suffix (String file_name) {
int index;
Node ret_file=null;


/*****************************************************************
Node Programming = create_node("Programming", "AN");
Programming.link(JAVA);
Programming.link(CLASS);
Node Text = create_node("Text", "AN");
Text.link(DOC);
Text.link(TXT);
Node Images = create_node("Images", "AN");
Images.link(JPG);
Images.link(PS);
*****************************************************************/


if (0 <= (index = file_name.toLowerCase().indexOf(".java"))) {
       ret_file = create_node("JAVA", "CN");
       }
else
if (0 <= (index = file_name.toLowerCase().indexOf(".class"))) {
       ret_file = create_node("CLASS", "CN");
       }
else
if (0 <= (index = file_name.toLowerCase().indexOf(".jpg"))) {
       ret_file = create_node("JPG", "CN");
       }
else
if (0 <= (index = file_name.toLowerCase().indexOf(".ps"))) {
       ret_file = create_node("PS", "CN");
       }
else
if (0 <= (index = file_name.toLowerCase().indexOf(".doc"))) {
       ret_file = create_node("DOC", "CN");
       }
else
if (0 <= (index = file_name.toLowerCase().indexOf(".txt"))) {
        ret_file = create_node("TXT", "CN");
        }
else
if (0 <= (index = file_name.toLowerCase().indexOf(".html"))) {
        ret_file = create_node("HTML", "CN");
        }
else
if (0 <= (index = file_name.toLowerCase().indexOf(".gif"))) {
        ret_file = create_node("GIF", "CN");
        }
else
if (0 <= (index = file_name.toLowerCase().indexOf(".c"))) {
        ret_file = create_node("C", "CN");
        }
return(ret_file);
} // end get_node_from_file_suffix


/**
 **  popf   FILES
 **
*/
public void popf () { // "show Files   or   Files"
Node file_node=null;


create_file_tree((File)null, 0);

file_node = datasea.find_node_named("Files");
datasea.clean(file_node);

GUI.P(0, "popf", "show Files   or   Files");
return;
} // end  popf



/**
 **  create_file_tree
 **
*/
public Node create_file_tree (File this_file, int current_depth) {
int i, max_length, index=0;
String list[], tName, path;
File tFile;
// user_dir_file=null; //user_dir_file is used temporarily only
char separator;
long mod, this_file_mod;
Node this_file_node, tFile_node, FilesNode=null;
boolean starting = false;
```

```
if (current_depth > MAX_DIRECTORY_DEPTH) // limit the depth of recursion
        return((Node)null);

FilesNode = create_node("Files", "AN");

if (this_file == null) { // if starting, link to DataSea.Root
        starting = true;
        GUI.P(0,"create_file_tree", "starting on user.home
<"+GUI.GlobalUserHomeDir+">");
        this_file = new File(GUI.GlobalUserHomeDir);         // Start from
UserDir,

        try {
                path = this_file.getCanonicalPath();
                GUI.P(0,"create_file_tree", "getCanonicalPath()="+path);
                }
            catch (IOException e) {
                GUI.P(0,"create_file_tree", "getCanonicalPath(): "+e);
                }
        }

this_file_node = create_node_forced(this_file.getName(), "DN", ""); // force
creation
this_file_node.isFile = true;

this_file_mod = this_file.lastModified();
separator = this_file.separatorChar;
path = this_file.getPath();


/*******************************************************
GUI.P(0,"create_file_tree", "path="+path);
GUI.P(0,"create_file_tree", "pathSeparator="+this_file.pathSeparator);
GUI.P(0,"create_file_tree", "separator="+this_file.separator);
GUI.P(0,"create_file_tree", "Name="+this_file.getName());
GUI.P(0,"create_file_tree", "Parent="+this_file.getParent());
GUI.P(0,"create_file_tree", "isDirectory="+this_file.isDirectory());
GUI.P(0,"create_file_tree", "====================================");
*******************************************************/

if (this_file.isDirectory()) {
        this_file_node.isDirectory = true;
        GUI.P(0,"create_file_tree","this File <"+this_file.getName()+"> is a
directory.");
        list = this_file.list();
        if (list == null) {
        GUI.WARNING(0,"create_file_tree","  list is null, returning.");
        return((Node)null);
        }
        max_length = (list.length > MAX_FILES_PER_DIRECTORY) ?
(MAX_FILES_PER_DIRECTORY) : (list.length);

        for (i=0; i< max_length; i++) {
                tName = path + separator + list[i];
                tFile = new File(tName);
                if (tFile != null) {
                        tFile_node = create_file_tree(tFile, current_depth+1);
                        if (tFile_node != null) {
                        mod = (this_file_mod - tFile.lastModified())/60000;// is
it in milliseconds? or even a valid time?
                        tFile_node.set_mag(Node.BARELY_VISIBLE_MAG);
                        this_file_node.link(tFile_node,
get_node_from_file_suffix(list[i]));
                        }
                        }
                }

        if (starting) {
                FilesNode.link(this_file_node);
                datasea.needdistUpdate = true;
                }
        }
return(this_file_node);
} // end create_file_tree

/**
 **  popn   NOTES
 **
*/
        public void popn () { // "show notes"
         int i;
        Node tn = datasea.find_node_named("Notes");
        if (tn==null) {
                GUI.ERROR(0, "", "node 'Notes' is null");
                return;
                }
GUI.input.string_input("input Notes This is a free-form note");
GUI.input.string_input("input Notes Tallis is red");
GUI.input.string_input("input Notes Tallis is a cat");
GUI.input.string_input("input Notes Tallis eats mice");
GUI.input.string_input("input Carl called about computer");
GUI.input.string_input("input Bob complained about desk");
GUI.input.string_input("input John Smith called about car");

GUI.input.string_input("input klavier:syn:piano");
GUI.input.string_input("input Bob's Klavier is German");
GUI.input.string_input("input Bob's piano is a Steinway");
GUI.input.string_input("input Bob mtg 4pm NextWeek");
GUI.input.string_input("input Bob mtg 4pm Oct 1999");
GUI.input.string_input("input Mtg with Jill 4pm Jun 3 2000");
GUI.input.string_input("input Mtg with IBMer 4pm Apr 3 2000");
GUI.input.string_input("input Bob is an IBMer");
GUI.input.string_input("input Jill is an IBMer");

//triplet("Carl","Red","HairColor");
//triplet("Carl","BMW","Car");
//GUI.input.string_input("input Carl's HairColor is Red");
//GUI.input.string_input("input Carl's Car is BMW");

GUI.input.string_input("input Mtg with Bob LastWeek");
GUI.input.string_input("input Mtg with Bob Today");
GUI.input.string_input("input Mtg with CommitteeX May 30 2000");

datasea.needdistUpdate = true;
GUI.P(0, "popn", "show Notes");
return;
} // end  (notes)


/**
 **  popx   EXTRA, MISC
 **
*/
          public void popx () { // "show web"
           int i;
          Node tn=null;

GUI.P(0,"", " Populate miscellaneous, Web & Files Run.");

          gen_array(Cluster_Web);

          Cluster_Files.link(gen_tree(Cluster_Files));

if ((tn = datasea.find_DN_named("D0.1.0")) != null)
          tn.link(datasea.cl("Music","AN"));
if ((tn = datasea.find_DN_named("D0.1.1")) != null)
          tn.link(datasea.cl("Music","AN"));
if ((tn = datasea.find_DN_named("D1.1.1")) != null)
          tn.link(datasea.cl("Music","AN"));

if ((tn = datasea.find_DN_named("D1.2.0")) != null)
          tn.link(datasea.cl("Theater","AN"));
if ((tn = datasea.find_DN_named("D1.2.1")) != null)
          tn.link(datasea.cl("Theater","AN"));
if ((tn = datasea.find_DN_named("D0.2.2")) != null)
          tn.link(datasea.cl("Theater","AN"));

GUI.P(0, "popx", "show Web");
} // end  popx
```

```
/**
 **  popmap
 **
*/
        public void popmap () { // "show map"
         int i;
         Node map_node, SF_node, LA_node, Berkeley_node;
         // create a map

map_node=new Node("Map","DN","Map of California", 100,100, 60,150);
LA_node = create_node("LA","DN");
SF_node = create_node("SF","DN");
Berkeley_node = create_node("Berkeley","DN");

LA_node.X = 10; LA_node.Y = -140;
SF_node.X = 10; SF_node.Y = 120;
Berkeley_node.X = 20; Berkeley_node.Y = 4;

map_node.link(LA_node);
LA_node.link(SF_node);
Berkeley_node.link(SF_node);

DataSea.Root.link(map_node);

GUI.P(0, "popmap", "show Map");
} // end popmap


/**
 **  popfab
 **
*/
        public void popfab () { // "show fab"
         int i;
         Node fab_node, X_node, Y_node, Z_node;
        Node node_1, node_2;
         // create a wafer fab data set with temp-machine data

GUI.P(0,"ab", " late VR, e.g. 'Fab' Run. ");
        fab_node=new Node("Fab","DN","Wafer Fab with machines and data",
100,100,600,400);
        X_node= new Node("Mach_X", "DN", "Machine X in wafer fab",0, -30,10,5);
        Y_node= new Node("Mach_Y", "DN", "Machine Y in wafer fab",50,-30,10,5);
        Z_node= new Node("Mach_Z", "DN", "Machine Z in wafer fab",90,-30,10,5);
fab_node.link(X_node);
fab_node.link(Y_node);
fab_node.link(Z_node);
(fab_node.getLinkTo(X_node)).setLinksVRparms(X_node);
(fab_node.getLinkTo(Y_node)).setLinksVRparms(Y_node);
(fab_node.getLinkTo(Z_node)).setLinksVRparms(Z_node);


node_1 = new Node("MachName","AN");
        node_1.link(X_node);
        node_1.link(Y_node);
        node_1.link(Z_node);

node_1 = new Node("Temp","AN");
node_2 = new Node("32","DN");
        node_1.link(node_2);
        X_node.link(node_2);
node_2 = new Node("12","DN");
        node_1.link(node_2);
        Y_node.link(node_2);
node_2 = new Node("21","DN");
        node_1.link(node_2);
        Z_node.link(node_2);


node_1 = new Node("Voltage","AN");
node_2 = new Node("v32","DN");
        node_1.link(node_2);
        X_node.link(node_2);
node_2 = new Node("v12","DN");
        node_1.link(node_2);
        Y_node.link(node_2);
node_2 = new Node("v21","DN");
        node_1.link(node_2);
        Z_node.link(node_2);

GUI.P(0, "popfab", "show Fab");
 datasea.needdistUpdate = true;
} // end popfab (VR)

/**
 ** popc
 **
*/
        public void popc () { // "show chromosome"
         int i;
         Node a,b,c,d;
         // create a chromosome with a few sites

        a=new Node("Chromosome","chromosome","Fake chromosome");
        DataSea.Root.link(a);

 datasea.needdistUpdate = true;
GUI.P(0, "popc", "show chrommosome");
} // end  (Chromosome)
```

```
/**
 **  DataSea.gen_array
 **
*/
public void gen_array (Node caller) {
       int i,j,k;
       Vector vec;
       Node tnode;
       int array_size = 20, x,y;

       Node CN3, CN4, CN5, CN6;

       vec = new Vector();
       for (i=0; i<array_size; i++) {
            x = (int)(caller.x +50-100*datasea.gui.random());
            y = (int)(caller.y +50-100*datasea.gui.random());
           tnode = new Node("el"+i, "DN", "from gen_array",
                 x,y);
           vec.addElement(tnode);
           caller.link(tnode);
           }

       CN3 = new Node("FirstHalf", "AN");
       for (i=0; i<array_size/2; i++)
              CN3.link((Node)vec.elementAt(i));

       CN4 = new Node("SecondHalf", "AN");
       for (i=array_size/2; i<array_size; i++)
              CN4.link((Node)vec.elementAt(i));

/**        CN5 = new Node("CN5", "DN");
       for (i=0; i<array_size; i+=5)
              CN5.link((Node)vec.elementAt(i));

       CN6 = new Node("CN6", "DN");
       for (i=0; i<array_size; i+=6)
              CN6.link((Node)vec.elementAt(i));
      caller.link(CN3);
      caller.link(CN4);
      caller.link(CN5);
      caller.link(CN6);
*/
       return;
}

/**
 **  gen_tree
 **
*/
```

```
        public Node gen_tree (Node caller) {
                int i,j,k;
         int branch_count = 4;
         int x, y;
                Node trunk, node_i, node_j, node_k;

         //x = (int)caller.x;
         //y = (int)caller.y;
        x = 0;
        y = 0;


                trunk = new Node("A", "DN", "Trunk of Tree",
                           0,0, 25,25);
                for (i=0; i<branch_count; i++) {
                        node_i = new Node("B"+i, "DN", "branch_node",
                          0, 0, 5, 5);
                        trunk.link(node_i);

                        for (j=0; j<branch_count; j++) {
                                node_j = new Node("C"+i+"."+j, "DN",
"branch_node",
                      0,0, 2,2);
                                node_i.link(node_j);

             for (k=0; k<branch_count; k++) {
                 node_k = new Node("D"+i+"."+j+"."+k, "DN", "leaf node",
                     0,0, 2,2);
                 node_j.link(node_k);
                if (k==0)
                        node_j.Type="DN";
                                                }
                                        }
                                }
                return(trunk);
} // end gen_tree



/**
 **  URLtoANpair    see if it exists, create it if need be, return it
 **
*/
        public Node URLtoANpair (String URL_name, String AN_name) {
         Node URL, AN;

//GUI.P(0,"URLtoANpair","Creating <"+URL_name+">, <"+AN_name+">");

        URL = create_node(URL_name, "URL", false);
        AN = create_node(AN_name, "AN", false);
        //URL.link(AN);
        URL.link(AN, Cluster_Web);

return(URL);
} // end URLtoANpair




/**
 **  URLtoURLpair    see if it exists, create it if need be, return it
 **
*/
        public Node URLtoURLpair (String URL1_name, String URL2_name) {
         Node URL1, URL2;

//GUI.P(0,"URLtoURLpair","Creating <"+URL1_name+">, <"+URL2_name+">");

        URL1 = create_node(URL1_name, "URL", false);
        URL2 = create_node(URL2_name, "URL", false);

        //URL1.link(URL2);
        URL1.link(URL2, Cluster_Web);

return(URL1);
} // end URLtoURLpair



/**
 **  ANpair    see if it exists, create it if need be, return it
 **
*/
        public Node ANpair (String AN1_name, String AN2_name) {
         Node AN1, AN2;

//GUI.P(0,"ANpair","Creating <"+AN1_name+">, <"+AN2_name+">");

        AN1 = create_node(AN1_name, "AN", false);
        AN2 = create_node(AN2_name, "AN", false);

        AN1.link(AN2);

return(AN1);
} // end ANpair
```

```
/**
 **  METApair    see if it exists, create it if need be, return it
 **
*/
        public Node METApair (String meta_name, String AN_name, String URL_name)
{
        Node meta, AN, URL;

GUI.P(0,"METApair","Creating meta name of <"+meta_name+">, value of
<"+AN_name+"> linked to URL "+URL_name);

        meta = create_node(meta_name, "PN", false);
        AN = create_node(AN_name, "AN", false);
        URL = create_node(URL_name, "URL", false);

        meta.link(AN);
        URL.link(AN);
        //meta.isPolarized = true;
        //AN.isPolarized = true;

datasea.gui.dump_node(0, true, meta);

return(AN);
} // end METApair



/**
 **  ANANpair    see if it exists, create it if need be, return it
 **
*/
        public Node ANANpair (String AN1_name, String AN2_name) {
        Node AN1, AN2;
        Node subnode;
        int num_words, i;
        String words[];

//GUI.P(0,"ANANpair","Creating <"+AN1_name+">, <"+AN2_name+">");

        AN1 = create_node(AN1_name, "AN", false);
        AN2 = create_node(AN2_name, "AN", false);

        AN1.link(AN2, "polarized");
        //AN1.isPolarized = true;
        //AN2.isPolarized = true;

// NOW SEE IF EITHER ARG CONSISTS OF INDIVIDUAL WORDS, MAKE UNPOLARIZED LINKS

        StringTokenizer t = new StringTokenizer(AN1_name, " ");
```

```
                num_words = t.countTokens();
                if (num_words != 1) { // if only one word, no need to break it apart
                words = new String[num_words];
                for(i = 0; i < num_words; i++) {
                        words[i] = t.nextToken();
                        subnode = create_node(words[i], "AN", false);
                        AN1.link(subnode, "unpolarized"); // unpolarized link
                        //GUI.P(0,"ANANpair","Linking AN1<"+AN1_name+">,
<"+subnode.Name+">");
                        }
                }

/****************************************************
                t = new StringTokenizer(AN2_name, " ");
                num_words = t.countTokens();
                if (num_words != 1) { // if only one word, no need to break it apart
                words = new String[num_words];
                for(i = 0; i < num_words; i++) {
                        words[i] = t.nextToken();
                        subnode = create_node(words[i], "AN", false);
                        AN2.link(subnode, "unpolarized"); // unpolarized link
                        //GUI.P(0,"ANANpair","Linking AN2<"+AN2_name+">,
<"+subnode.Name+">");
                        }
                }
****************************************************/

return(AN1);
} // end ANANpair


/**
 **  ANDNpair    see if it exists, create it if need be, return it
 **
*/
        public Node ANDNpair (String AN1_name, String DN2_name) {
        Node AN1, DN2;

//GUI.P(0,"ANDNpair","Creating <"+AN1_name+">, <"+DN2_name+">");

        AN1 = create_node(AN1_name, "AN", false);
        DN2 = create_node(DN2_name, "DN", false);

        AN1.link(DN2);
        //AN1.isPolarized = true;
        //DN2.isPolarized = true;

return(AN1);
} // end ANDNpair
```

```
/**
 **  triplet    see if it exists, create it if need be, return it
 **                 Automatically create a matching AN for a DN
*/
        public Node triplet (String DN1_name, String DN2_name, String AN_name) {
         Node DN1, DN2, AN;

return(triplet(DN1_name, DN2_name, AN_name, null, null));
} // end triplet


/**
 **  triplet    see if it exists, create it if need be, return it
 **                 Automatically create a matching AN for a DN
*/
        public Node triplet (String DN1_name, String DN2_name, String AN_name,
                                  Node CNode, String polarized_string) {
         Node DN1, DN2, AN;
        Link link;

GUI.P(1,"triplet","Creating <"+DN1_name+">, <"+DN2_name+">, <"+AN_name+">");

        DN1 = create_node(DN1_name, "DN", false);
        DN2 = create_node(DN2_name, "DN", false);
        AN  = create_node(AN_name, "AN", false);

                // this way to polarize from DN1->DN2 and AN->DN2
        DN1.link(DN2, CNode, polarized_string);
        AN.link(DN2, CNode, polarized_string);

        link = DN1.getLinkTo(DN2);
        if (link != null)
                link.Name = AN.Name;

//      AN.isCN = true;
//      link.addCNode(AN); // have AN modulate link between DN1 and DN2
//      link = DN2.getLinkTo(AN);
//      link.addCNode(AN); // have AN modulate links to itself

        //AN.isPolarized = true;
        //DN1.isPolarized = true;
        //DN2.isPolarized = true;
return(DN1);
} // end triplet



/**
 **  make_BB
 **
*/
public Node make_BB (String cmd, String Name) {
int i, size;
Node newBBnode;
//String type = "BB";
String type = "DN";

if (cmd.equalsIgnoreCase("start")) {
        lastBBnode = create_node_forced(Name,type,"");
        GUI.P(0,"make_BB","Created node "+lastBBnode.Name+",
Type="+lastBBnode.Type);
        }
else if (cmd.equalsIgnoreCase("add")) {
        newBBnode = create_node_forced(Name,type,"");
        lastBBnode.link(newBBnode);
        GUI.P(0,"make_BB","Created node "+newBBnode.Name+",
Type="+newBBnode.Type+", linked to "+lastBBnode.Name);
        lastBBnode = newBBnode; // save this one for later calls
}

return(lastBBnode);

} // end make_BB



/**
 **  create_node    Name only, ForceCreation=false
 **
*/
        public Node create_node (String Name) {
        return( create_node(Name, "DN", "", false) );
        }

/**
 **  create_node    Name and Type only, ForceCreation=false
 **
*/
        public Node create_node (String Name, String Type) {
        return( create_node(Name, Type, "", false) );
} //

/**
 **  create_node     Name, Type and ForceCreation only
 **
*/
        public Node create_node (String Name, String Type, boolean ForceCreation)
{
        return( create_node(Name, Type, "", ForceCreation) );
} //
/**
 **  create_node    Name, Type and Desc only, ForceCreation=false
 **
*/
        public Node create_node (String Name, String Type, String Desc) {
        return( create_node(Name, Type, Desc, false) );
} //  end create_node

/**
 **  create_node    Full version: see if it exists, create it if need be (or is
forced),
 **                 return it
 **                 (Automatically create a matching AN for a DN if >1 DN's of
same name)
*/
        public Node create_node (String Name, String Type, String Desc, boolean
ForceCreation) {
        Node ret_node=null, tnode=null;



        if (ForceCreation) {
        datasea.gui.P(0, "create_node","ForceCreate  is
true!!!!!!!!!!!!!!!!!!!!");
                if (Type.equals("AN")) {// If forcing an AN, make top-level then
a child
                        tnode = datasea.find_node_named(Name, Type);
                        if (tnode == null) { // get top-level AN, then add another
AN
                                tnode = create_node_forced(Name,Type,Desc);
                                }
                        ret_node = create_node_forced(Name,Type,Desc);
                        tnode.link(ret_node);
                        return(ret_node);
                        }
                else
                return(create_node_forced(Name,Type,Desc));
                }
        else // don't ForceCreation
        if ((ret_node=datasea.find_node_named(Name))==null)  {
                ret_node = new Node(Name, Type, Desc);
                }
                else {
                if (ret_node.Desc.equals(""))
                        ret_node.Desc = Desc;
                }

return(ret_node);
} //  end create_node
```

```
/**
 **  create_node_forced
 **
*/
public Node create_node_forced (String Name, String Type) {

return(create_node_forced(Name, Type, ""));

} // end create_node_forced


/**
 **  create_node_forced
 **
*/
public Node create_node_forced (String Name, String Type, String Desc) {
int i, size;
Node ret_node=null, other_dn_node=null, an_node=null;

if (Type.equalsIgnoreCase("DN")) {
       // see if there's an AN and link it to the new DN

 // if there's a DN already, make sure an AN exists and link them
       if (null != (other_dn_node=datasea.find_node_named(Name, "DN"))) {
              an_node = datasea.find_node_named(Name, "AN");
              if (an_node == null) {
                     an_node = create_node(Name, "AN", false);
                     other_dn_node.link(an_node);
                     }
              }
       ret_node = new Node(Name, Type, Desc); // forced
       ret_node.link( an_node ); // may or may not exist
       }
else
       ret_node = new Node(Name, Type, Desc); // forced

return(ret_node);
} // end create_node_forced



/**
 **  get
 **
*/
public void get_a_URL (Node node) {
int i, size=0;
Node child;

System.err.println("============= get_a_URL  ... Begun
...=================================");
System.err.println("============= "+node.Name+",
Type="+node.Type+"====================");

if (node.isURL) {
        System.err.println("============= Calling pull_in_URLs on
"+node.Name+"==========================");
        pull_in_URLs(node.Name, "", 1, (Node)null);
        }
else { // do it on all the children
        System.err.println("============= Calling pull_in_URLs on children of
"+node.Name+"==========================");
        size = node.Links.size();
        for (i=0; i<size; i++) {
                child = node.getNodeAtLink(i);
                if (child.isURL && (child.dist > node.dist))
                        pull_in_URLs(child.Name, "", 1, (Node)null);
                }
        }

} // end get_a_URL


} // end Populate
```

```
// This is GUI.java     by Rocky Nevin

import java.applet.*;
import java.lang.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.sql.*;  // for the class Timestamp
import java.io.*;


/**
 * This is GUI.java     by Rocky Nevin
 * which is the top-level object for DataSea, instantiated within this.
 *
LeftButton
MiddleButton     -> Alt down
RightButton      -> Meta down

 *
 * @version     0.1, 8/10/98
 * @version     0.2, 9/6/98
 * @version     0.3, 11/4/98
 *  run() -> update() ->   [ paint(Node, layer_index) calls
position_and_render_start(Node caller, int dist_to_POV)]
 * @version     0.4, 3/12/98
 *
 *
 *  Principal Methods:
    add_buttons
    demo1/2
    dump
    init
    main
    position_start/child_node/recursive
    render_start/recursive
    run_thread
    sleep
    update, paint
...
2/6/99
The bottom functions (render/position) return boolean whether to recurse,
assuming a VR function handles things in node-dependent ways.

Node.VRyes, via VRyes(Node), individually allows setting of VR mode
render_start()
        |-> render_recursive() lastly calls render_recursive()
            |-> render_node()

position_start()
```

```
        |-> position_recursive() lastly calls position_recursive()
            ||-> position_node()

NEED: to correct nsr_position_XXXXX functions, e.g. _DIR, _DIR_ENTRY

...
2/25/99
Focus on TimeLine, storing POV's and not recursing through them (spread, etc)

6/1/99
shift, meta etc are key events, handled in key_input(), and may also
be queried in processEvent (either newButton or newFrame) by
checking (InputEvent)(event).isShift(), isMeta(), etc.

6/6/99
Remember:
        width depends on node.size_x which is constant and in data coordinates.
        in VRmode, if node.x depends on another node, its node.x=other.x(+-
)other.size_x/2
        magscale and WindowOffsets occur only once, in drawing on screen, i.e.
using map() fn
//   height = (int)((child.size_y));
                That is, node.x|X|dx|dX is in data space and should thus not
depend on viewing parms

6/7/99
Very odd: I don't see why the old style of POV or Parent pressure
brings in remote high-value mags, e.g. from 'sim', but it does.
Probably the force is simply strongly negative, the direction tending
vertically from POV to its linked node.
Note also that 2pm:today: is not linked to Today
Note also 'text' for EMF is not placed nicely
Note large data text isn't handled.

6/8/99
Fixed the thread interaction problem (I think), whereby unlinking nodes while a
thread was animating, and insize a recursive rendering routine, would give
invalid
references to pointers.
Changed child positioning to be fan-shaped, POV pressure now segregates higher
mag
distal children.
Application EMail now automatically resets view and POV and zooms on the email
form.
Looks good.

6/11/99
Consider way to have parsed input linked distally to ANs, like email, phone,
etc.
6/12/99
```

```
Problem with TL position with getLinksVRparms() is that I've not called
setLinksVRparms()
on all of them, therefore VRyes() returns false often.
6/17/99
Added vote_branch, to vote on branches
Need to add time-dependent display, e.g. pulsating mags to see effects, 'what
if's
6/18/99
Added draw_string() which wraps text, and started popEcon().
Showing TL, then resetting and showing again, it's children Today, Tomorrow,
Yesterday
are in different positions, same X though, same linkage as before.
6/21/99
Need abs to be more general, e.g. spreads through anything, not just DNs
TL and friends is a mess, not positioned as expected.
6/27/99
Inhibited recursion if dist==-1, added Node.importance

END OF COMMENTS
 */
public class GUI extends Applet implements Runnable {

int INFINITE_DEPTH = 1000; // just a number to give essentially infinite depth
of levels
int image_counter = 1;
public static int counter_of_positioned_nodes = 0;
public static int max_transition_count = 1;
public static Graphics graphics;
public static Graphics graphics1, graphics2;
public static Graphics graphics_frame2;
public static Graphics text_graphics;
public static Frame frame, frame2, text_frame, diag_frame;
Image image1, image2;
static Mode mode_obj;
static String GlobalUserDir;
static String GlobalUserHomeDir;
static String GlobalOSName;
static boolean quick = true;
static boolean doNormalization = true;
static boolean showBoundaries = false;
static boolean checkPolarization = true;
static boolean auto_rescale = true;
static boolean parse = true;
static double spread_factor=0.15;
static double thetaOffset=0.4;
static double max_x, min_x, max_y, min_y; // used by rescale() and auto_rescale
static int desired_visible_count = 30; // used by DataSea.auto_flatten()
8/24/99
Point GlobalMapPoint; //
static Point TinyPoint; // used by map()
```

```
Force force;
int drawing_counter=0;
static boolean Animating = false;
static boolean NetOK = true;
static boolean shrinking_allowed = true;
static boolean is_meta_down = false;
static boolean is_alt_down = false;
static boolean is_control_down = false;
static boolean is_shift_down = false;
static boolean coordinates_on=true;
static boolean drawBoxes = true;
static boolean TinyFlag = false;
static boolean globalDoTiny = false;
static boolean global_ok_to_draw = false;
static double global_angle_array[];
//static double TinyScale = 1.0;
static double globalMaxPressure = 1.0;
static boolean StopThreadRequest = false;
static boolean FlipAxes = false;
static boolean meta=false;
static boolean alt=false;
static boolean shift=false;
static boolean control=false;
static int counter=0;
static int mouseX=0;
static int mouseY=0;
static int spaceX=0;
static int spaceY=0;
static double magscale = 1.0;
double pressure_mag = 3.0;
static int updateCount = 8;
static int Globaldist;
static double DEFAULT_TEXT_THRESHOLD=Node.BIG_MAG;
static double DEFAULT_POS_THRESHOLD=Node.BARELY_INVISIBLE_MAG;
static double DEFAULT_RELATIONS_THRESHOLD=Node.MED_MAG;
static double DEFAULT_BOX_THRESHOLD=Node.MED_MAG;
static double pos_threshold          = Node.BARELY_INVISIBLE_MAG;
static double relations_threshold    = Node.MED_MAG;
static double text_threshold         = Node.MED_MAG-1;
static double box_threshold          = Node.MED_MAG;
static Thread animation_thread;
static GUI gui;
static DataSea datasea;
static Input input;
static Timer timer;
static TL tl;
static int recursion_depth = 1;
static int dist_limit = 100;
static boolean drawText = true;
static boolean drawLinkNames = true;
static boolean drawFile = false;
static boolean drawDirectory = true;
static boolean drawWind = true;
static boolean drawAN = true;
static boolean drawON = true;
static boolean drawDN = true;
static boolean drawEvent = true;
static boolean drawCN = false;
static boolean drawPN = true;
static boolean drawURL = true;
static Node Xnode=null;
static Node lastNode=null;
static Node TinyNode = null;
static Node SavedNode = null;
static Node Myself; // This is a self-node, one which can be used to show the
DataSea
                    // program itself
static Node VR_MACH_NODE, VR_FAB_NODE;
boolean string_active = false;
String TitleString = "";
String key_input_string = "";
static String lastCommand = "";
static String StatusLine[];
static String global_str[]=null;
static int global_str_size=0;
static int MAX_GLOBAL_STR_SIZE=140;
Timestamp timestamp;
java.util.Date date;
static java.lang.Double java_lang_double;
static java.lang.Long java_lang_long;
// Node node;
static long current_TS;
static long thisCommandTS;
static long lastCommandTS;
static int Debug = 0;
static boolean Details = true;
static int GlobalNodeNode = 0;
static Font titleFont;
static Font littleFont;
Font buttonFont;
// static int WindowWidth=600, WindowHeight=600;
static int WindowWidth=1240, WindowHeight=1028; // defaults, set in init()
static int WindowXcenter=WindowWidth/2, WindowYcenter=WindowHeight/2;
static int WindowXOffset=0, WindowYOffset=0;
static String priorCommand = " ";
static Vector selected_nodes_vec; // holds all nodes that are selected
public static KeyEvent ke;
TextField text_field;

static List list;
```

```
MyDialog query_dialog, input_dialog;
static newButton button_animate;
static newButton button_reset;
static newButton button_absorb_POV;
static newButton button_populate;
static newButton button_position;
static newButton button_background_color;
static newButton button_render;
static newButton button_lines;
static newButton button_potentiation;
static newButton button_postprocessor;
static newButton button_increase_mag;
static newButton button_Debug;
static newButton button_Dump;
static newButton button_more_attraction;
static newButton button_more_repulsion;
static newButton button_Stop;
static newButton button_drawText;
static newButton button_drawDN;
static newButton button_drawCN;
static newButton button_presParent;
static newButton button_presPOV;
static newButton button_presNeighbors;
static newButton button_presNoise;
static Checkbox check;
static double ThetaMultiplier=Math.PI/2;


ColorObj color_obj;


public static void main (String[] argv)
{
gui=new GUI();
}

public GUI() {
super();
Graphics[] graphicsarray;
if (gui == null)
init();
}



/**
 **  beep
 **
*/
static public void beep () {
```

```
int i, size;
Node tn;

Toolkit.getDefaultToolkit().beep();

} // end beep

/**
 * If G is the top level object of an applet, init() gets run automatically.
 * Linked objects and animation
 */
public void init () { // frame, graphics, font
int i;

if (gui == null)
       gui = this;

dump_properties();

global_str = new String[MAX_GLOBAL_STR_SIZE];
StatusLine = new String[15];
StatusLine[0] = "--------------------";
for (i=0; i<15; i++)
       StatusLine[i] = "";

Toolkit.getDefaultToolkit().beep();
System.out.println(Toolkit.getDefaultToolkit().getScreenSize());
WindowWidth=Toolkit.getDefaultToolkit().getScreenSize().width;
WindowHeight=Toolkit.getDefaultToolkit().getScreenSize().height;
WindowXcenter=WindowWidth/2;
WindowYcenter=3*WindowHeight/5;

timer = new Timer();
input = new Input(gui);
color_obj = new ColorObj();
color_obj.init();
reset_current_TS();
date = new java.util.Date();
timestamp = new Timestamp(date.getTime());
frame = new newFrame("This is a Frame");
frame.setSize( WindowWidth, WindowHeight);
Point p = new Point(20,0);
frame.setLocation( p );
// frame.setBackground(Color.white);
frame.setBackground(color_obj.DarkGrey);
//frame.setBackground(color_obj.VeryLightGrey);
frame.setLayout(new java.awt.FlowLayout()); // 8/22/99
//frame.setLayout(new java.awt.BorderLayout());
frame.setTitle("DataSea: "+date.toString());
```

```
TitleString = "DATASEA: CONFIDENTIAL AND PROPRIETARY Information of Rocky Nevin,
"+date.toString();
frame.setTitle(TitleString);
//frame.show();
//frame.setVisible(true);
graphics = frame.getGraphics();
status("Hello, this is GUI.init() running. OS_Name="+GlobalOSName+",
UserDir="+GlobalUserDir);
titleFont = new java.awt.Font("Arial", Font.BOLD, 12);
littleFont = new java.awt.Font("Helvetica", Font.BOLD, 8);
frame.setFont(titleFont);
frame.setVisible(true);



text_frame = new Frame("URL Frame");
text_frame.setSize( 200, 400 );
p = new Point(WindowWidth-200, 0);
text_frame.setLocation( p );
text_frame.setBackground(color_obj.VeryLightBlue);
// text_frame.setLayout(new java.awt.BorderLayout());
// TitleString =
// text_frame.setTitle(TitleString);
// text_frame.show();
// text_graphics = text_frame.getGraphics();

diag_frame = new Frame("Diagnostic Messages");
diag_frame.setSize( WindowWidth, 150 );
p = new Point(0, WindowHeight - 130);
diag_frame.setLocation( p );
diag_frame.setBackground(Color.black);
diag_frame.setBackground(color_obj.VeryLightGreen);
list = new List(22);
list.setEnabled(true);
diag_frame.add("North", list);

mode_obj = new Mode();
force = new Force();
GlobalMapPoint = new Point();
java_lang_double = new Double(0);
java_lang_long = new Long(0);

add_buttons();
image1 = frame.createImage(WindowWidth, WindowHeight);
image2 = frame.createImage(WindowWidth, WindowHeight);
graphics1 = image1.getGraphics();
graphics2 = image2.getGraphics();


datasea = new DataSea(this);  // Construct Data Sea
```

```
input = new Input(this);       // Construct Data Sea
tl = new TL(); // TimeLine object


datasea.pop.populate_begin();


// test();

run_thread();

datasea.word_reset();
global_ok_to_draw = true;

} // end init


/**
 ** write_to_frame2
 */
public void write_to_frame2 (String s) {

graphics_frame2.clearRect(0,0, (frame2.getSize()).width,
(frame2.getSize()).height);


date = new java.util.Date();
TitleString = "DATASEA: CONFIDENTIAL AND PROPRIETARY Information of Rocky Nevin,
"+date.toString();

        graphics_frame2.setColor(Color.blue);
         graphics_frame2.drawString(date.toString(), 10,40);

        graphics_frame2.setColor(Color.black);
         graphics_frame2.drawString("abcdefg counter = "+counter, 10,100);
         graphics_frame2.drawString(s, 10,110);
return;
} // end write_to_frame2


/**
 **  extra_frame
 **
*/
        public void extra_frame () {
frame2 = new newFrame("This is a Frame");
frame2.setSize( 600, 400);
frame2.setLocation( new Point(0,(int)(WindowHeight*0.8)) );
frame2.setBackground(color_obj.VeryLightGrey);
frame2.setLayout(new java.awt.BorderLayout());
frame2.setTitle("Frame2: "+date.toString());
```

```
frame2.setVisible(true);
graphics_frame2 = frame2.getGraphics();
sleep(1000); // need to wait before writing to it
} // end extra_frame


public void test () {
int c;
char b[];
b = new char[1000];



extra_frame();
write_to_frame2("Hello, this is a test from extra_frame()");
// try {
       File inputFile = new File("/usr/people/rocky");
       // FileInputStream fis = new FileInputStream(inputFile);
       // while ( (c=fis.read(b)) != -1)
       //      System.err.print("!!:>>>"+b[0]+b[1]+b[2]+b[3]+b[4]+b[5]+"<<<");
       // fis.close();
       //
       // FileReader fr = new FileReader(inputFile);
       file_dump(inputFile);
       // }
       // catch (FileNotFoundException e) { ERROR(0,"test","File
/usr/people/rocky/x not found."); }
       // catch (IOException e) { ERROR(0,"test","IOException."); }

calc_angles();


/*************************************
* FontMetrics fontMetrics;
* fontMetrics = new FontMetrics(titleFont);
* System.err.println("FontMetrics.charsWidth()="
*       +fontMetrics.charsWidth(chrs,0,10));
*************************************/

/*****************
* P(0,"init","theta(1,0)="+get_angle(1,0));
* P(0,"init","theta(1,1)="+get_angle(1,1));
* P(0,"init","theta(0,1)="+get_angle(0,1));
* P(0,"init","theta(-1,0)="+get_angle(-1,0));
* P(0,"init","theta(-1,-1)="+get_angle(-1,-1));
* P(0,"init","theta(0,-1)="+get_angle(0,-1));
***************/

/*****************
```

```
* Runtime rt = Runtime.getRuntime();
* System.err.println("init(): Runtime freeMemory is "+rt.freeMemory());
* try { rt.exec("date"); }
*     catch (java.io.IOException e) { System.err.println("Runtime IOException:
"+e.toString()); }
* // rt.traceMethodCalls(true);
***************/


} // end test


/**
 **  file_dump
 **
*/
public void file_dump (File inputFile) {
int i;
String list[], tName;
File tFile;
char separator;
separator = inputFile.separatorChar;
long mod;

P(0,"file_dump", "pathSeparator="+inputFile.pathSeparator);
P(0,"file_dump", "separator="+inputFile.separator);
P(0,"file_dump", "Name="+inputFile.getName());
P(0,"file_dump", "Parent="+inputFile.getParent());
P(0,"file_dump", "isDirectory="+inputFile.isDirectory());
list = inputFile.list();
P(0,"file_dump", ""+list.length+" entries ...");
for (i=0; i< list.length; i++) {
        tName = inputFile.getPath() + separator + list[i];
        tFile = new File(tName);
        mod = tFile.lastModified();
        mod /= 60000;
        if (tFile.isDirectory())
                P(0,"file_dump", i+")Dir: "+tName+", last modified "+mod+" min's
ago");
        else
                P(0,"file_dump", i+")File: "+tName+", last modified "+mod+" min's
ago");
        }

return;
} // end file_dump

/**
 **  calc_angles
 **
```

```
*/
        static public void calc_angles () {
        double theta=0;
         int i;
        double x, y;
        long TS1=0, TS2=0;

timer.start_timer("nothing");
timer.end_timer("nothing");

timer.start_timer("allocating 10000 doubles");
        global`angle_array = new double[10000];
timer.end_timer("allocating 10000 doubles");


TS1 = java.lang.System.currentTimeMillis();
TS2 = java.lang.System.currentTimeMillis();
System.err.println("Time of nothing is "+(TS2-TS1)+" milliseconds");

timer.start_timer("get angles");
TS1 = java.lang.System.currentTimeMillis();
        for (i=0; i<10000; i++) {
                theta = get_angle((double)i,(double)1.3);
        }
TS2 = java.lang.System.currentTimeMillis();
System.err.println("Time of 10000 get_angle's  is "+(TS2-TS1)+" milliseconds");
timer.end_timer("get angles");


TS1 = java.lang.System.currentTimeMillis();
        for (i=0; i<10000; i++) {
                theta = Math.atan2((double)i,(double)1.3);
        }
TS2 = java.lang.System.currentTimeMillis();
System.err.println("Time of 10000 atan2's  is "+(TS2-TS1)+" milliseconds");


TS1 = java.lang.System.currentTimeMillis();
        for (i=0; i<10000; i++) {
                theta = i * (2*Math.PI / 10000.0);
                global_angle_array[i] =  Math.sin(theta);
        }
TS2 = java.lang.System.currentTimeMillis();
System.err.println("Time of 10000 sin(x) calc's into array is "+(TS2-TS1)+"
milliseconds");

// Time doing a thousand sine's
TS1 = java.lang.System.currentTimeMillis();
        for (i=0; i<10000; i++) {
                theta = i * (2*Math.PI / 10000.0);
                x =  Math.sin(theta);
                if (TS1>TS2)
                        y = x;
        }
TS2 = java.lang.System.currentTimeMillis();
System.err.println("Time of 10000 sin(x) is "+(TS2-TS1)+" milliseconds");

// Time retrieving a thousand elements
TS1 = java.lang.System.currentTimeMillis();
System.err.println("Time of printing above "+(TS1-TS2)+" milliseconds");
TS1 = java.lang.System.currentTimeMillis();
        for (i=0; i<10000; i++) {
                theta = i * (2*Math.PI / 10000.0);
                x =  global_angle_array[i];
                if (TS1>TS2)
                        y = x;
        }
TS2 = java.lang.System.currentTimeMillis();
System.err.println("Time of 10000 elements of global_angle_array is "+(TS2-
TS1)+" milliseconds");



System.err.println("global_angle_array[20]="+global_angle_array[20]);
System.err.println("global_angle_array[120]="+global_angle_array[120]);
System.err.println("global_angle_array[220]="+global_angle_array[220]);
} // end calc_angles

/**
 **  dump_properties
 **
*/
public void dump_properties () {
GlobalUserDir = System.getProperty("user.dir");
GlobalUserHomeDir = System.getProperty("user.home");
GlobalOSName = System.getProperty("os.name");

System.err.println(" ");
System.err.println("==========================================================
");
System.err.println("version:           "+System.getProperty("java.version"));
System.err.println("vendor:            "+System.getProperty("java.vendor"));
System.err.println("vendor.url:
        "+System.getProperty("java.vendor.url"));
System.err.println("class.version:
        "+System.getProperty("java.class.version"));
System.err.println("os.name:           "+System.getProperty("os.name"));
System.err.println("os.arch:           "+System.getProperty("os.arch"));
System.err.println("file.separator:    "+System.getProperty("file.separator"));
System.err.println("user.name:                 "+System.getProperty("user.name"));
System.err.println("user.home:              "+System.getProperty("user.home"));
System.err.println("user.home:              "+GlobalUserHomeDir);
System.err.println("user.dir:           "+System.getProperty("user.dir"));

// System.err.println("class.path:    "+System.getProperty("java.class.path"));
// System.err.println("path.separator:
       "+System.getProperty("path.separator"));
// System.err.println("line.separator:
       "+System.getProperty("line.separator"));

System.err.println("========================================================
");
System.err.println(" ");


return;
} // end dump_properties



/**
 ** clear
 */
public void clear () {
       int x1, y1,  x2, y2;

       graphics.clearRect(0,0, (frame.getSize()).width,
(frame.getSize()).height);
       if (coordinates_on) {
       graphics.setXORMode(color_obj.LightRed);
       map(-100, 0,GlobalMapPoint); x1 = GlobalMapPoint.x; y1 =
GlobalMapPoint.y;
       map(100, 0,GlobalMapPoint); x2 = GlobalMapPoint.x; y2 = GlobalMapPoint.y;
       graphics.drawLine(x1, y1, x2, y2);

       map(0, 100,GlobalMapPoint); x1 = GlobalMapPoint.x; y1 = GlobalMapPoint.y;
       map(0, -100,GlobalMapPoint); x2 = GlobalMapPoint.x; y2 =
GlobalMapPoint.y;
       graphics.drawLine(x1, y1, x2, y2);
       graphics.setPaintMode();
       }
} // end clear



/**
 **  amplify_region
 **
 */
```

```
        public void amplify_region(int val) {
        P(0,"amplify_region","Got val="+val+", no action coded yet.");
} // end amplify_region


/*
 * key_input
 * this handles key input, if key is Enter, then accumulated
 * string 'GUI.key_input_string' is given to input.string_input()
 */
public void key_input (KeyEvent ke) {
    int key = ke.getKeyCode();
    String keystring = ke.getKeyText(key);
    char ch = ke.getKeyChar();
  P(2,"key_input","Keystring is <"+keystring+">, KeyChar is <"+ch+">");

    if (keystring.equals("Shift")) {
        }
    else if (keystring.equals("Alt")) {
        }
    else if (keystring.equals("Ctrl")) {
        }
    else
    if (keystring.equals("Enter")) {
        input.string_input(key_input_string);
        if (!key_input_string.equals("u")) {
               lastCommand = key_input_string;
               p("lastCommand: "+lastCommand);
               }
        key_input_string = "";
        status(lastCommand);
        }
// HANDLE PLUS KEY and MINUS KEY  plus key minus key
    else if ((ch == '+') && (key_input_string.equals("")) && (lastNode != null))
{
        lastNode.more_mag();
        datasea.normalize();
        }
    else if ((ch == '-') && (key_input_string.equals("")) && (lastNode != null))
{
        dump_node(0, false, lastNode);
        lastNode.less_mag();
        datasea.normalize();
        dump_node(0, false, lastNode);
        }
    else if ( keystring.equals("Up") || keystring.equals("Down")
            || keystring.equals("Left") || keystring.equals("Right") )
            arrow_key(keystring);
    else if ( keystring.equals("F1") || keystring.equals("F2")
                || keystring.equals("F3") || keystring.equals("F4")
                || keystring.equals("F5") || keystring.equals("F6") )
                F_key(keystring);
        else if ( keystring.equals("Backspace") ) {
            if ( key_input_string.equals(""))
                    ;
            else {
                    key_input_string = key_input_string.substring(0, -
1+key_input_string.length());
                    }
            }
        else {
            key_input_string = key_input_string+ch;
              }

} // end key_input

/**
 ** F_key(String)
 */
 public void F_key (String str) {

     if (str.equals("F1")) {
         P(0,"F_key","F1: Help:");
         help();
         }

     if (str.equals("F2")) {
         if (lastNode == null)
             WARNING(0,"F_key", "F2: need a selected 'lastNode' to link to POV");
         else {
             P(0,"F_key","F2: linking POV to "+lastNode.Name);
             input.string_input("show "+lastNode.Name);
             }
         }
     if (str.equals("F3")) {
         if (lastNode==null)
             WARNING(0,"F_key", "F3: need a selected 'lastNode' to find common
nodes");
         else {
             P(0,"F_key","F3: stimulate similar nodes to "+lastNode.Name);
             datasea.find_common_nodes_to(lastNode);
             }
         }
      if (str.equals("F4")) {
         P(0,"F_key","F4: change POV");
         if (lastNode != null)
                datasea.POV = lastNode;
         }
      if (str.equals("F5")) {
            P(0,"F_key","F5: unlink all and link lastNode to POV");
            solo_link_to_POV();
            }
        if (str.equals("F6")) {
            P(0,"F_key","F6: unlink lastNode");
            if (datasea.POV!=null)
                    datasea.POV.unlink_both(lastNode);
            else
                    WARNING(0,"word_selector","Need a POV to work.");
            }


return;
} // end F_key



/**
 **  solo_link_to_POV
 **
*/
        public void solo_link_to_POV() {
         int i;
         Node a, b, c, d;
        if (datasea.POV == null) {
                WARNING(0,"solo_link_to_POV","Need a POV to work.");
                if (lastNode != null)
                        datasea.show(lastNode);
                return;
                }
        if (lastNode == null) {
                WARNING(0,"solo_link_to_POV","Need a lastNode to work.");
                return;
                }
        stop_thread();
        datasea.POV.unlink_all();
        datasea.POV.link(lastNode);
        datasea.needdistUpdate = true;
        //if (animation_thread == null) {
        //      run_thread();
        //      }
} // end solo_link_to_POV

/**
 ** èhelp
 */
 public void help () {
    P(0,"help"," n toggles NeighborPressure,  l toggles lines");
    P(0,"help"," 'abs [ ]' magnifies abstractions of a DN, 'back [ ]' mag's
backwards to POV, 'most'=create POV and link to greatest mags");
```

```
    P(0,"help"," 'ren arg1 [ ]'=rename, 'link arg1 [ ]', 'unlink arg1 [ ]', 'del
[ ]'=delete node");
    P(0,"help"," 'rr|rv' chooses rendering relations or VR, 'l' toggles lines,
'n' toggles neighbor pressure 'select', 'unselect'(all)");
    P(0,"help"," 'focus [ ]'=make a node the POV, 'le', 'ri', 'up', 'do' =
left,right,up,down;  'ttup', 'ttdown' = text_threshold up or down by 0.1");
    P(0,"help","popd,TL,f,m,n,s,c,x,VR,  da|dd=draw CN|DNs");
    P(0,"help","F4=turn lastNode into POV  F5=unlink_all from POV and link
lastNode to POV  F6=unlink lastNode from POV");
    P(0,"help","F1=help  F2=connect lastNode to POV  F3=show similar nodes to
lastNode");
        return;
 }




/**
 **  shift_one_node
 **
*/
public void shift_one_node (Node node, String direction) {
int i, size, dist_diff=1;
Node child, saved_node=null;
double saved_mag;

if (node == null)
       return;

saved_mag = 0;
if (direction.equalsIgnoreCase("distal"))
       dist_diff = 1;
if (direction.equalsIgnoreCase("proximal"))
       dist_diff = -1;


size = node.Links.size();
for (i=0; i<size; i++) {
       child = node.getNodeAtLink(i);
       if ((child.dist == (node.dist+dist_diff)) && node!=datasea.Root) { //
check the direction
       if (child.mag > saved_mag) { // save the biggest candidate
              saved_mag = child.mag;
              saved_node = child;
              }
       }
}

if (saved_node != null) {
        P(0,"shift_one_node","direction '"+direction+"',  "+lastNode.Name+" ->
"+saved_node.Name);
        lastNode = saved_node;
        if (lastNode.mag < Node.MED_MAG)
                lastNode.mag = Node.MED_MAG;
        }

} // end shift_one_node


/**
 ** arrow_key(String)
 */
 public void arrow_key (String str) {

if (lastNode != null) {
if (is_control_down) {
    if (str.equals("Up"))
        lastNode.Y += 10;
    if (str.equals("Down"))
        lastNode.Y -= 10;
    if (str.equals("Right"))
        lastNode.X += 10;
    if (str.equals("Left"))
        lastNode.X -= 10;
        P(0,"arrow_key", " ->For lastNode.Name="+lastNode.Name+" got "+str+",
changing node.X|Y");
        }
else {
    if (str.equals("Up"))
        shift_one_node(lastNode, "distal");
    if (str.equals("Down"))
        shift_one_node(lastNode, "proximal");
    if (str.equals("Right"))
        ;
    if (str.equals("Left"))
        ;
        }
    }
else { // no lastNode available
        if (Debug == 2)
                P(2,"arrow_key", " ->Got "+str+ " NO lastNode, shifting entire
screen.");
    if (str.equals("Up"))
            WindowYOffset += 100/magscale;

    if (str.equals("Down"))
            WindowYOffset -= 100/magscale;

    if (str.equals("Right"))
```

```
                WindowXOffset += 100/magscale;

            if (str.equals("Left"))
                WindowXOffset -= 100/magscale;

        }
    }

    /**
     **   handle_mouse_clicked
     **   @params  Node node, the node already determined to be under the mouse
     **   Find the nearest node to the cursor.
    */
    public void handle_mouse_clicked (Node node) {


    lastNode = node;
    P(1,"handle_mouse_clicked","is_meta_down="+is_meta_down+ ",
    is_alt_down="+is_alt_down+ ", is_control_down="+is_control_down+ ",
    is_shift_down="+is_shift_down);

    // control keys are down ...
            if (GUI.control)
                    solo_link_to_POV();
            else if (GUI.alt)
                    action(node);
            else if (GUI.shift)
                    datasea.vote_branch(null, "+");
            else if (is_meta_down)    // Right mouse button
                    datasea.inhibit(node);
            else { // Left mouse button
                    dump_node(0, false, node);
                    ; // just set the node as lastNode
            }
    /*****************************
    action (alt)
    POV (ctl)
    vote - (meta ... Rmouse)
    vote + (shift)
    select (plain)
    ***************************/

    datasea.normalize();

    return;
    } // end handle_mouse_clicked

    /**
     **   mouse_clicked_at
     **   @params  int x,y, the cursor position where the mouse is.
```

```
 **  Find the nearest node to the cursor.
*/
public Node mouse_clicked_at (int x, int y) {
    Node node=null, tnode=null;
    double tdist, saved_dist = 10000000;
    int i, size;

// Search through all nodes and find nearest one to where mouse was clicked
    size = datasea.node_vec.size();
    for (i=0; i<size; i++) {
     tnode = (Node)datasea.node_vec.elementAt(i);
       if (       (tnode.mag >= Node.SMALL_MAG)
               &&((!tnode.isURL || drawURL)
               && (!tnode.isAN || drawAN)
               && (!tnode.isFile || drawFile)
               && (!tnode.isDN || drawDN))) {
               tdist = (tnode.x-x)*(tnode.x-x) + (tnode.y+tnode.size_Y/2-
y)*(tnode.y+tnode.size_Y/2-y);
               if (tdist < saved_dist) {
               saved_dist = tdist;
               node = tnode;
               }
       }
    }
lastCommand = "Mouse selected '"+node.Name+"', mag="+node.mag+",
d="+node.dist+", ChldCnt="+node.ChildCount;
handle_mouse_clicked(node);
return (node);
} // end mouse_clicked_at

/*
** newFrame instead of Frame
*/
class newFrame  extends Frame {
newFrame(String name) {
                super(name);
                enableEvents(
                      AWTEvent.ACTION_EVENT_MASK
                //    | AWTEvent.ADJUSTMENT_EVENT_MASK
                //    | AWTEvent.COMPONENT_EVENT_MASK
               //     | AWTEvent.CONTAINER_EVENT_MASK
                //    | AWTEvent.FOCUS_EVENT_MASK
                //    | AWTEvent.ITEM_EVENT_MASK
                      | AWTEvent.KEY_EVENT_MASK
                      | AWTEvent.MOUSE_EVENT_MASK
                      | AWTEvent.MOUSE_MOTION_EVENT_MASK
               //     | AWTEvent.TEXT_EVENT_MASK
               //     | AWTEvent.WINDOW_EVENT_MASK
                      );
}
```

```
// Odd, I can't get KEY_PRESSED events from newFrame's processEvent, even with
KEY_EVENT_MASK on

        public void processEvent (AWTEvent event) {   // newFrame
    int x, y;
    Node tn;


if (event.getID() == MouseEvent.MOUSE_MOVED) {
// ALL THIS IS TO GET COORDINATES OF MOUSE
         x =((MouseEvent)event).getX();
         y =((MouseEvent)event).getY();
        GUI.mouseX=x;
        GUI.mouseY=y;
         x -= WindowXcenter;
         y -= WindowYcenter;
         x /= magscale;
         y /= magscale;
         x += WindowXOffset;
         y -= WindowYOffset;
        GUI.spaceX=x;
        GUI.spaceY=-y;
        if (Debug == 4)
                P(4,"processEvent.newFrame",
                "(newFrame )event='" +event.paramString()+
                "', X="+((MouseEvent)event).getX()+
                ", id="+event.getID());
        }

// How to determine if it's a MouseEvent or WindowEvent?
// Resizing the window apparently sends resize message to a button,
//          and the button becomes the size of the window.
/*************************************************************
** if (event.getID() != MouseEvent.MOUSE_MOVED) {
**          P(0,"processEvent.newFrame", "toString()="+event.toString());
**          P(0,"processEvent.newFrame", "paramString()="+event.paramString());
**          P(0,"processEvent.newFrame", "getID()="+event.getID());
** }
** if (
** (event.getID() != WindowEvent.WINDOW_ACTIVATED)   &&
** (event.getID() != WindowEvent.WINDOW_DEACTIVATED)  &&
** (event.getID() != WindowEvent.WINDOW_CLOSING)  &&
** (event.getID() != WindowEvent.WINDOW_CLOSED)  &&
** (event.getID() != WindowEvent.WINDOW_OPENED)  &&
** (event.getID() != WindowEvent.WINDOW_CLOSED)  &&
** (event.getID() != WindowEvent.WINDOW_ICONIFIED)&&
** (event.getID() != WindowEvent.WINDOW_DEICONIFIED)
** )
*************************************************************/
        {  // THIS IS NEVER RUN ...
```

```
            is_meta_down = ((InputEvent)event).isMetaDown();
             is_alt_down = ((InputEvent)event).isAltDown();
             is_control_down = ((InputEvent)event).isControlDown();
            if (is_control_down)
                    System.out.println("---------- is_control_down ------------------
----- xxxx");
             is_shift_down = ((InputEvent)event).isShiftDown();
             GUI.meta = is_meta_down;
             GUI.alt = is_alt_down;
             GUI.shift = is_shift_down;
             GUI.control = is_control_down;
             if (is_control_down)
                P(0,"newFrame.processEvent","Control down!!!");
             }
            if (Debug==4) {
             if (is_meta_down)
                P(0,"newFrame.processEvent","Meta down!!!");
             if (is_alt_down)
                P(0,"newFrame.processEvent","Alt down!!!");
             if (is_control_down)
                P(0,"newFrame.processEvent","Control down!!!");
             if (is_shift_down)
                P(0,"newFrame.processEvent","Shift down!!!");
             }
// ------------------------------------------------
// ------------------------------------------------
             if (event.getID() == MouseEvent.MOUSE_CLICKED)
             {
             tn=mouse_clicked_at(GUI.spaceX, GUI.spaceY);     // I draw Y increasing
going up
             }
          } // End method processEvent()
}   // End class newFrame

class newButton extends Button {
        String original_name;

        newButton(String name) {
                super(name);
                original_name = name;  // Set original name
                enableEvents(
                        AWTEvent.ACTION_EVENT_MASK
                        | AWTEvent.ADJUSTMENT_EVENT_MASK
                        | AWTEvent.COMPONENT_EVENT_MASK
                        | AWTEvent.CONTAINER_EVENT_MASK
                        | AWTEvent.FOCUS_EVENT_MASK
                        | AWTEvent.ITEM_EVENT_MASK
                        | AWTEvent.KEY_EVENT_MASK
                        | AWTEvent.MOUSE_EVENT_MASK
                        | AWTEvent.MOUSE_MOTION_EVENT_MASK
```

```
                              | AWTEvent.TEXT_EVENT_MASK
                     //       | AWTEvent.WINDOW_EVENT_MASK
                              );
               }

               public void processEvent (AWTEvent event) {  // newButton
                     long TS = 0;
                     Button b,bb;
                     double val;


               if (Debug==4)
                P(4,"processEvent.newButton",
                     "(newButton )event='" +event.paramString()+
                     ", id="+event.getID());


                     if (event.getID() == KeyEvent.KEY_PRESSED) {
                            KeyEvent keyevent = (KeyEvent)event;
                            int key = keyevent.getKeyCode();
                            String keystring = keyevent.getKeyText(key);
                            char ch = keyevent.getKeyChar();
                            key_input(keyevent);
          // HERE
          //System.out.println("PRESSED keystring("+keystring+") keycode("+key+")
          keyevent("
          //     +event.toString()+")");
                            if (key == 16)
                                   is_shift_down = true;
                            if (key == 17)
                                   is_control_down = true;
                            if (key == 18)
                                   is_alt_down = true;
                            }
                     if (event.getID() == KeyEvent.KEY_RELEASED) {
                            KeyEvent keyevent = (KeyEvent)event;
                            int key = keyevent.getKeyCode();
                            String keystring = keyevent.getKeyText(key);
                            char ch = keyevent.getKeyChar();
                            //key_input(keyevent);   DON'T DO IT ON KEY_RELEASED, IT
          SCREWS UP INPUT
          // HERE
          //System.out.println("RELEASED keystring("+keystring+") keycode("+key+")
          keyevent("
          //     +event.toString()+")");
                            if (key == 16)
                                   is_shift_down = false;
                            if (key == 17)
                                   is_control_down = false;
                            if (key == 18)
```

```
                        is_alt_down = false;
                    }
                if (event.getID() == MouseEvent.MOUSE_CLICKED) {
                    TS = java.lang.System.currentTimeMillis();
                    boolean is_meta_down = ((InputEvent)event).isMetaDown();
        if (is_meta_down)
         if (Debug==4)
            P(4,"newButton.processEvent","Meta down!!!");

                    if (this == button_animate) {

                    debug_animation_thread(1,"buttons",
"this==button_animate");
                    if (animation_thread == null) {
                        run_thread();
                        } else {
                        stop_thread();
                        }
                    }
                    if (this == button_presParent)
                        {
                 force.calcParentPressure = !force.calcParentPressure;
                graphics.setColor(Color.red);
                 this.setLabel("Parent pres'="+force.calcParentPressure);
                if (force.calcParentPressure)
                    button_presParent.setBackground(color_obj.LightRed);
                else
                    button_presParent.setBackground(color_obj.LightGrey);
                 }
            if (this == button_presPOV)
                        {
                 force.calcPOVPressure = !force.calcPOVPressure;
                graphics.setColor(Color.blue);
                 this.setLabel("POV="+force.calcPOVPressure);
                if (force.calcPOVPressure)
                    button_presPOV.setBackground(color_obj.LightRed);
                else
                    button_presPOV.setBackground(color_obj.LightGrey);
                 }
                    if (this == button_presNeighbors)
                        {
                 force.calcNeighborPressure = !force.calcNeighborPressure;
                // graphics.setColor(Color.green);
                 this.setLabel("Nbr="+force.calcNeighborPressure);
                if (force.calcNeighborPressure)
                    button_presNeighbors.setBackground(color_obj.LightRed);
                else
                    button_presNeighbors.setBackground(color_obj.LightGrey);
                 }
                    if (this == button_presNoise)
```

```
                              {
                        force.calcNoise = !force.calcNoise;
                        this.setLabel("Noise="+force.calcNoise);
                        }
                              if (this == button_increase_mag) {
                                     }
                              if (this == button_drawText) {
                                     mode_obj.toggle_draw_text();
                                     }
                              if (this == button_absorb_POV) {
                                     datasea.absorb_POV(false);
                                     }
                              if (this == button_postprocessor) {
                                     datasea.PostProcessor();
                                     }
                              if (this == button_more_attraction) {
                                     val=datasea.more_attraction(is_meta_down);
                                     this.setLabel("Mag Scale="+val);
                                     }
                              if (this == button_more_repulsion) {
                                     val=datasea.more_repulsion(is_meta_down);
                                     this.setLabel("ThetaMlt="+val);
                                     }
                              if (this == button_lines) {
                                     mode_obj.toggle_lines_mode();
                                     }
                              if (this == button_drawCN) {
                                      drawCN = !drawCN;
                                      this.setLabel("CN="+drawCN);
                                     }
                              if (this == button_drawDN) {
                                     drawDN = !drawDN;
                                     this.setLabel("DN="+drawDN);
                                     }

// First time, create Dialog window. Subsequently create new node

if (this == button_background_color) {    // SPREADING MODE
       color_obj.background_color_index++;
       if (color_obj.background_color_index >= color_obj.MAX_BACKGROUND_COLORS)
              color_obj.background_color_index=0;
       frame.setBackground(color_obj.background_color_array[color_obj.background
_color_index]);
       show_buttons();
              }
if (this == button_position) {    // POSITIONING MODE
              mode_obj.toggle_position_mode();
              }
if (this == button_render) {    // RENDERING MODE
               mode_obj.toggle_render_mode();
```

```
                    }
        if (this == button_potentiation) {    // POTENTIATION MODE
                    mode_obj.toggle_do_potentiation();
                    }
                   if (this == button_Dump) {
                              recursion_depth=0;
                             dump_nodes(0, datasea.Root);  // arg is
debug_value

                             // dump_node(0, Myself);  // arg is debug_value
                             }
                       if (this == button_Debug) {

// See if left or right button is pressed
                             if (is_meta_down)
                                 Debug --;
                             else
                                 Debug ++;

// Set max/min for Debug
                             if (Debug > 5)
                                 Debug = 5;
                             if (Debug < -1)
                                 Debug = -1;
                             this.setLabel("Debug="+Debug);
                             }

            if (this == button_Stop) {
                    quit();
                             }
                         }
                   super.processEvent(event);
                   }
}  // class NewButton

/**
**
*/
class MyDialog extends Dialog {
    TextField   text;
    public MyDialog(Frame f, String t) {
        super(f, t);
        setSize(200, 100);
       text = new TextField("Hi, MyDialog creation.", 150);
       }
}


public void quit () {
try { java.lang.System.exit(0); }
```

```
    catch (SecurityException e) { ; }
} // end quit

/**
 *
 *
 */
        void add_buttons () {
        int x=80, y=20;

        int i = 0;
         frame.add(button_animate = new newButton("    Start    "));
         button_animate.setBackground(color_obj.LightGrey);
         button_animate.setForeground(Color.black);
         button_animate.setLocation(90*i, 30);
         button_animate.setSize(x, y);
         button_animate.setVisible(true);

        i++;
         frame.add(button_Stop = new newButton("EXIT"));
         button_Stop.setBackground(Color.pink);
         button_Stop.setForeground(Color.black);
         button_Stop.setLocation(90*i, 30);
         button_Stop.setSize(x,y);
         button_Stop.setVisible(true);

        i++;
         frame.add(button_Debug = new newButton("Debug="+Debug+" "));
         button_Debug.setBackground(color_obj.LightGrey);
         button_Debug.setForeground(Color.black);
         button_Debug.setLocation(90*i, 30);
         button_Debug.setSize(x,y);
         button_Debug.setVisible(true);

        i++;
         frame.add(button_render = new newButton("Node Rendering"));
         button_render.setBackground(color_obj.LightGrey);
         button_render.setForeground(Color.black);
//          button_render.setLocation(90*i, 30);
         button_render.setSize(x,y);
         button_render.setVisible(true);

        i++;
         frame.add(button_background_color = new newButton("Background Colors"));
         button_background_color.setBackground(color_obj.LightGrey);
         button_background_color.setForeground(Color.black);
         button_background_color.setLocation(90*i, 30);
         button_background_color.setSize(x+30,y);
         button_background_color.setVisible(true);
        i++;
         frame.add(button_position = new newButton("Pos'"));
         button_position.setBackground(color_obj.LightGrey);
         button_position.setForeground(Color.black);
         button_position.setLocation(90*i, 30);
         button_position.setSize(x,y);
         button_position.setVisible(true);
        i++;
         frame.add(button_potentiation = new newButton("Pot
"+mode_obj.do_potentiation));
        if (mode_obj.do_potentiation)
                button_potentiation.setBackground(color_obj.LightRed);
        else
               button_potentiation.setBackground(color_obj.LightGrey);
         button_potentiation.setForeground(Color.black);
         button_potentiation.setLocation(90*i, 30);
         button_potentiation.setSize(x,y);
         button_potentiation.setVisible(false);
        i++;
         frame.add(button_Dump = new newButton("---- Dump Nodes----"));
         //button_Dump.setBackground(new Color(0xffbbbb));
         button_Dump.setBackground(color_obj.LightGrey);
         button_Dump.setForeground(Color.black);
         button_Dump.setLocation(90*i, 30);
         button_Dump.setSize(x,y);
         button_Dump.setVisible(false);
         frame.add(check = new Checkbox("CheckBox."));
         check.setVisible(false);


        i=0;
         frame.add(button_drawText = new newButton("Inhibit Text"));
        if (drawText)
                button_drawText.setBackground(color_obj.LightRed);
        else
               button_drawText.setBackground(color_obj.LightGrey);
         button_drawText.setForeground(Color.black);
         button_drawText.setLocation(90*i, 60);
         button_drawText.setSize(x,y);
         button_drawText.setVisible(true);

        i++;
         frame.add(button_lines = new newButton("Lines=      "));
         button_lines.setBackground(color_obj.LightGrey);
         button_lines.setForeground(Color.black);
         button_lines.setLocation(90*i, 60);
         button_lines.setSize(x,y);
         button_lines.setVisible(true);
        i++;
        frame.add(button_presParent = new newButton("Parent
Pressure="+force.calcParentPressure));
        if (force.calcParentPressure)
               button_presParent.setBackground(color_obj.LightRed);
        else
               button_presParent.setBackground(color_obj.LightGrey);
        button_presParent.setLocation(90*i, 60);
        button_presParent.setForeground(Color.black);
        button_presParent.setSize(x,y);
        button_presParent.setVisible(true);
        i++;
        frame.add(button_presPOV = new newButton("POV="+force.calcPOVPressure));
        if (force.calcPOVPressure)
               button_presPOV.setBackground(color_obj.LightRed);
        else
               button_presPOV.setBackground(color_obj.LightGrey);
        button_presPOV.setForeground(Color.black);
        button_presPOV.setLocation(90*i, 60);
        button_presPOV.setSize(x,y);
        button_presPOV.setVisible(true);
        i++;
        frame.add(button_presNeighbors = new
newButton("NeighborPressure="+force.calcNeighborPressure));
        if (force.calcNeighborPressure)
               button_presNeighbors.setBackground(color_obj.LightRed);
        else
               button_presNeighbors.setBackground(color_obj.LightGrey);
        button_presNeighbors.setForeground(Color.black);
        button_presNeighbors.setLocation(90*i, 60);
        button_presNeighbors.setSize(x,y);
        button_presNeighbors.setVisible(true);
/*****************************************************************
**         i++;
**         frame.add(button_drawCN = new newButton("CN="+drawCN));
**         button_drawCN.setBackground(color_obj.LightGrey);
**         button_drawCN.setForeground(Color.black);
**         button_drawCN.setLocation(90*i - 65, 60);
**         button_drawCN.setSize(x/2,y);
**         button_drawCN.setVisible(false);
**         i++;
**         frame.add(button_drawDN = new newButton("DN="+drawDN));
**         button_drawDN.setBackground(color_obj.LightGrey);
**         button_drawDN.setForeground(Color.black);
**         button_drawDN.setLocation(90*(i-1), 60);
**         button_drawDN.setSize(x/2,y);
**         button_drawDN.setVisible(false);
**         frame.add(button_absorb_POV = new newButton("Absorb POV"));
**         button_absorb_POV.setBackground(color_obj.LightGrey);
**         button_absorb_POV.setForeground(Color.black);
**         button_absorb_POV.setLocation(90*i, 60);
**          button_absorb_POV.setSize(x,y);
**          button_absorb_POV.setVisible(false);
*********************************************************************
*********************************************************************
**              i++;
**          frame.add(button_postprocessor = new newButton("PostProcessor"));
**          button_postprocessor.setBackground(color_obj.LightGrey);
**          button_postprocessor.setForeground(Color.black);
**          button_postprocessor.setLocation(90*i, 60);
**          button_postprocessor.setSize(x,y);
**          button_postprocessor.setVisible(false);
**      i++;
**          frame.add(button_reset = new newButton("Reset"));
**          button_reset.setBackground(color_obj.LightGrey);
**          button_reset.setForeground(Color.black);
**          button_reset.setLocation(90*i, 60);
**          button_reset.setSize(x,y);
**          button_reset.setVisible(false);
**          i++;
**          frame.add(button_populate = new newButton("Populate."));
**          button_populate.setBackground(color_obj.LightGrey);
**          button_populate.setForeground(Color.black);
**          button_populate.setLocation(90*i, 60);
**          button_populate.setSize(x,y);
**          button_populate.setVisible(false);
**          i++;
*********************************************************************
*********************************************************************
**          frame.add(button_more_attraction = new newButton("Screen Mag."));
**          button_more_attraction.setBackground(color_obj.LightGrey);
**          button_more_attraction.setForeground(Color.black);
**          button_more_attraction.setLocation(90*i, 30);
**          button_more_attraction.setSize(x,y);
**          button_more_attraction.setVisible(false);
**      i++;
**          frame.add(button_more_repulsion = new newButton("ThetaMlt"));
**          button_more_repulsion.setBackground(color_obj.LightGrey);
**          button_more_repulsion.setForeground(Color.black);
**          button_more_repulsion.setLocation(90*i, 30);
**          button_more_repulsion.setSize(x,y);
**          button_more_repulsion.setVisible(false);
**      i++;
*********************************************************************
*********************************************************************
**      i++;
**          frame.add(button_presNoise = new
newButton("Noise="+force.calcNoise));
**          button_presNoise.setBackground(color_obj.LightGrey);
**          button_presNoise.setForeground(Color.black);
**          button_presNoise.setLocation(90*i, 90);
**          button_presNoise.setSize(x,y);
**          button_presNoise.setVisible(false);
*********************************************************************/


        mode_obj.set_mode_buttons();
}



/**
 **  show_buttons
 **
*/
public void show_buttons () {
System.err.println("show_buttons.");
        button_animate.setBackground(color_obj.LightGrey);
        button_animate.setForeground(Color.black);
        button_Debug.setBackground(color_obj.LightGrey);
        button_Debug.setForeground(Color.black);
        button_render.setBackground(color_obj.LightGrey);
        button_render.setForeground(Color.black);
        button_background_color.setBackground(color_obj.LightGrey);
        button_background_color.setForeground(Color.black);
        button_Stop.setBackground(color_obj.LightGrey);
        button_Stop.setForeground(Color.black);
        button_drawText.setBackground(color_obj.LightGrey);
        button_drawText.setForeground(Color.black);
        button_lines.setBackground(color_obj.LightGrey);
        button_lines.setForeground(Color.black);
        button_presParent.setBackground(color_obj.LightGrey);
        button_presParent.setForeground(Color.black);
        button_presPOV.setBackground(color_obj.LightGrey);
        button_presPOV.setForeground(Color.black);
        button_presNeighbors.setBackground(color_obj.LightGrey);
        button_presNeighbors.setForeground(Color.black);
} // end show_buttons


/*
 * map   returns a Point in screen coordinates for rendering, considering
 *                                             WindowOffsets and magscale
 */
public void map (int x, int y, Point MapPoint) {
    map((double)x, (double)y, MapPoint);
    return;
}
public void map (double x, double y, Point MapPoint) {
double tx, ty;
double temp_x=0, temp_y=0;
```

```
tx = x - WindowXOffset;
tx *= magscale;
tx += WindowXcenter;
ty = y - WindowYOffset;
ty *= magscale;
ty = WindowYcenter - ty; // Y IS FLIPPED


// Now we have normal coordinates, consider TinyFlag
// get diff' with embedded-universe node (e.g. TimeLine), compress and add back
to TL.x's
// if (TinyFlag && globalDoTiny)
/***********************************************************
if (TinyFlag) {
        if (TinyPoint != null) {
                temp_x = tx-TinyPoint.x;
                temp_y = ty-TinyPoint.y;
                temp_x *= TinyScale;
                temp_y *= TinyScale;
                tx = TinyPoint.x + temp_x;
                ty = TinyPoint.y + temp_y;
                }
        }
*******************************************************/

if (FlipAxes) {
        MapPoint.y = (int)tx;
        MapPoint.x = (int)ty;
        }
else {
        MapPoint.x = (int)tx;
        MapPoint.y = (int)ty;
        }


return;
} // end map



/*
**
*
*
void clear_Tdist (Node parent){
Node tnode;
int i, size;

    size = datasea.node_vec.size();
    for (i=0; i<size; i++) {
        tnode = (Node)datasea.node_vec.elementAt(i);
        tnode.Tdist = -1;
        }
return;
} // end clear_Tdist

/*
**
*
*
void clear_dist (Node parent){
Node tnode;
int i, size;

    size = datasea.node_vec.size();
    for (i=0; i<size; i++) {
        tnode = (Node)datasea.node_vec.elementAt(i);
        tnode.dist = -1;
        }
return;
} // end clear_dist
***********************************************/


/*
*  VRyes    simple function to return whether to use VR mode or heirarchical
mode
*/
static boolean VRyes (Node node) { // position this parent's children
boolean node_yes=false;

if (node == null) {
        ERROR(0,"VRyes", "Null node passed to me.");
        return(false);
        }

if ((node.X != 0) || (node.Y != 0))
        node_yes = true;
else if (node == datasea.POV)
        node_yes = true;

if (node.VRyes || (mode_obj.render_mode.equals("VR") && node_yes))
        return(true);
else
        return(false);
} // end VRyes


/**
```

```
 **  draw_global_str
 **
*/
public void draw_global_str () {
int i;
Color current_color=null;

// DRAW GLOBAL STRING IF AVAILABLE

if (global_str_size > 0)
{
current_color = graphics.getColor();
graphics.setColor(color_obj.LightGrey);
graphics.setColor(Color.black);
// Limit printed results to 40 lines, else can get horrible slow
global_str_size = (global_str_size>40) ? 40 : global_str_size;
for (i=0; i<global_str_size; i++)  {
   if (global_str[i] != null)
      graphics.drawString( global_str[i], 30,350+11*i );
   else
      ERROR(0, "draw_global_str", "NULL node global_str["+i+"]");
}

graphics.setColor(current_color);
}

} // end draw_global_str


/**
 **  clear_status
 **
*/
public void clear_status () {
int i;
StatusLine[0] = "--------------------";
for (i=0; i<15; i++)
       StatusLine[i] = "";
} // end clear_status


/**
 **  draw_status
 **
*/
public void draw_status () {
int i;
Color current_color=null;

// DRAW STATUS LINE STRING
```

```
current_color = graphics.getColor();
//graphics.setColor(color_obj.LightBlue);
graphics.setColor(Color.black);

graphics.drawString( "nodes drawn: "+counter_of_positioned_nodes, 10, 68);
graphics.drawString( "frame:"+drawing_counter++, 20, 80);

graphics.drawString("prior command:", 10,90);
graphics.drawString("<"+lastCommand+">", 20, 102);
graphics.drawString("current command:", 10, 114);
graphics.drawString(key_input_string, 20, 126);

graphics.drawString( "Command History:", 10, 150);

for (i=0; i<15; i++) {
        if (!StatusLine[i].equals(""))
               draw_string(StatusLine[i], 20, 160+12*i);
               //graphics.drawString( StatusLine[i], 20, 160+12*i);
}
graphics.setColor(current_color);

return;
} // end draw_status

/*
 *
 */
synchronized float position_start (Node parent){

if (parent == null)
   return(-1);

if (DataSea.needdistUpdate) {
        datasea.set_dist_start(parent);
        datasea.set_ChildNum_start((Node)null, parent, datasea.theta_org);//false
for theta_org_flag
        datasea.needdistUpdate = false;
        }

        position_recursive((Node)null, parent); // POSITION

return(0);
} // end position_start (a simple function calling recursive fns)


/**
 **  rescale
 **
*/
public void rescale (String cmd) {
```

```
int i, size;
Node node;
Node child;
double new_magscale, magdif;

if (cmd.equalsIgnoreCase("clear")) {
max_x =  20; // we want to always include the (0,0) cross-hairs
min_x = -20;
max_y =  20;
min_y = -20;
counter_of_positioned_nodes = 0;
}
else if (cmd.equalsIgnoreCase("run")) {
double range_x = max_x - min_x;
double range_y = max_y - min_y;
double magx, magy;

WindowWidth=frame.getSize().width;
WindowHeight=frame.getSize().height;
WindowXcenter = WindowWidth/2;
WindowYcenter = WindowHeight/2;

        magx = WindowWidth/range_x;
        magy = WindowHeight/range_y;
        new_magscale = (magx<magy) ? magx : magy;
// Slowly change offsets ...
        WindowXOffset += (int)(0.2*(((min_x + max_x)/2.0)-WindowXOffset));
        WindowYOffset += (int)(0.2*(((min_y + max_y)/2.0)-WindowYOffset));
        new_magscale *= 0.75; // Show all the edges

// Now, slowly change magscale
        magdif = (new_magscale-magscale);
        if (magdif > 0.5)      // don't allow it to zoom-in quickly
                magdif = 0.5;
        if (magdif < -1.60)    // allow it to pull back quickly
                magdif = -1.60;
        magscale += magdif;

/************************************************************************
 P(1,"rescale, run","min_x("+min_x+"), max_x("+max_x+"),magx="+magx+",  magscale
= "+magscale+", WindowXOffset = "+WindowXOffset);
 P(1,"rescale, run","min_y("+min_y+"), max_y("+max_y+"),magy="+magy+",  magscale
= "+magscale+", WindowYOffset = "+WindowYOffset);
 P(1,"rescale, run","WindowXcenter="+WindowXcenter+", Y="+WindowYcenter);
************************************************************************/
}

} // end rescale
```

```
/**
 **  random   I have a feeling the seed is being reused ... 9/99  no, not true
 **
*/
static public double random () {
//return(Math.random() + ((double)(java.lang.System.currentTimeMillis()) %
10.0)/10.0);
return(Math.random());
} // end random


/**
 **  flash
 **
*/
public void flash (Node node) {


double x=random(), y=random();
Point TempPoint=new Point();
Color current_color=null;

if (!datasea.gui.global_ok_to_draw)
        return;

if (node==null)
        return;

node.isSelected = true;
show_node_once(node);
node.isSelected = false;
sleep(30);
/*******************************************************
x += node.x;
y += node.y;
map(x,y, TempPoint);

graphics.setXORMode(Color.green);
//current_color = graphics.getColor();
//graphics.setColor(Color.red);
graphics.drawLine(TempPoint.x, TempPoint.y, TempPoint.x+2, TempPoint.y); // draw
a fake chromosome
//graphics.setColor(current_color);
graphics.setPaintMode();
*******************************************************/

} // end flash
```

```
/**
 **  OKtoRenderRecursive
 **
*/
public boolean OKtoRenderRecursive (Node parent, Node child) {
boolean continue_flag=false;

if (GUI.mode_obj.position_mode.equals("Lvls") ||
       GUI.mode_obj.position_mode.equals("Grid"))
       return(true);

if (parent == datasea.POV)
       return(true);
if (child == datasea.POV)
       return(true);
if (child.dist <= 4)
       return(true);

// Make sure we should render child, based on dist, then recurse
if (child == null) {
   WARNING(0, "GUI.OKtoRenderRecursive", "NULL CHILD, RETURNING.
parent="+parent.Name+"(dist="
     +parent.dist+"), child="+child.Name+"(dist="+child.dist+")");
   return(false);
   }

// Don't draw more than once per 'paint()'
if (child.drawnTS == current_TS) {
       return(false);
       }
else
       child.drawnTS = current_TS;

if (child.dist == -1)    { // We'll never see this or its children if its not
linked to POV
       return(false);
       }

if (child.isEvent && drawEvent)
    continue_flag |= true;
if (child.isCN && drawCN)
    continue_flag |= true;
if (child.isAN && drawAN)
    continue_flag |= true;
if (child.isON && drawON)
    continue_flag |= true;
if (child.isDN && drawDN)
    continue_flag |= true;
if (child.isFile && drawFile)
```

```
    continue_flag |= true;
if (child.isURL && drawURL)
    continue_flag |= true;

if (!continue_flag)
        return(false);
if (child.mag < relations_threshold && !child.isSelected) {
        return(false);
        }
if (parent != null && child.isPOV)  {  // Don't draw towards a POV
        return(false);
        }
if ((parent != null) && (child == datasea.Root))  {// don't draw Root as
anyone's child
        return(false);
        }
return(true);
} // end OKtoRenderRecursive



/*
 *
 */
float render_start (Node parent){

if (parent == null)
   return(-1);

clear();
graphics.setColor(color_obj.LightGrey);
if (global_str_size > 0)
        draw_global_str();
draw_status();
render_recursive(null, parent); // DRAW

return(0);
} // end render_start (a simple function calling recursive fns)


/*
 * render_recursive   checks VRyes(child) and calls rendering fn's, then
recurses
 */
public void render_recursive (Node parent, Node child) { //
int i, size;
Node grand_child, CNode;
Point ChildPoint=new Point();
double prior_max_mag=0;
boolean PolOK = false;
boolean OK=true;

OK = OKtoRenderRecursive(parent, child);

if (quick && !OK) {
        return;
        }

// temp override
        color_obj.set_color_from_TS(graphics, child);
        render_node(parent, child, false);
/**************************************************************
if (xor)
        graphics.setXORMode(Color.white);
else
if (child.isSelected == true)
        graphics.setColor(Color.red);
else
        color_obj.set_color_from_mag(graphics, child);
**************************************************************/

// MAKE RECURSIVE CALLS ON CHILDREN
size = child.Links.size();
for (i=0; i<size; i++) { // check dist of all children, position if appropriate

// This is dumb, but I have trouble controling threads and synchronizing blocks
/**************************************************************/
        if (child == null) // bad synchronization between rendering and linking
threads
                break;
        if (i >= child.Links.size()) // bad synchronization between rendering and
linking threads
                break;
/**************************************************************/
        grand_child = child.getNodeAtLink(i);

// 4/29/2000
        if (grand_child == null)
                continue;

if (OK) { // don't draw if not OK, but do recurse nonetheless

// 4/29/2000
        if (quick) {
        if (grand_child.mag < Node.MED_MAG &&  child.mag < Node.MED_MAG) {
                //System.out.print(" R["+grand_child.Name+"] ");
                continue;
                }
        }
```

```
// See if we should render link-modulating CN's
       if (drawCN) {
       CNode = child.getCNodeAtLink(i);
       if (CNode != null) {
              //P(2,"render_recursive",
              //      "["+i+"] CNode of "+child.Name+" to "+grand_child.Name+"
is "+CNode.Name);
              render_node(child, CNode, false);
              }
       }
} // if OK
if (grand_child.hasLargerDistThan(child)) { // recurse regardless of OK
              render_recursive(child, grand_child);
       }
} // for i
return;
} // end render_recursive


/*
 *
 */
void position_recursive (Node parent, Node child) { // position this parent's
children
int i, size;
boolean PolOK = false;
Node grand_child, CNode=null;


   if (child == null) {
       WARNING(0,"position_recursive", "child is null");
       return;
       }

if (child.dist < 1) {
       //WARNING(0,"position_recursive", "child dist <1"+child.Name);
       return;
       }


   //size = child.Links.size();
   size = child.ChildCount;
   for (i=0; i<size; i++) { // position and then recurse on all children
       // ordered list, set in set_ChildNum_recursive()
        grand_child = child.getNodeAtLink(child.child_vec[i]);
       if (grand_child == null) {
              return;
              }
       if (grand_child.isAN && !drawAN) {
              return;
```

```
                }
        if (grand_child.isEvent && !drawEvent) {
                return;
                }
// 4/29/2000
        if (quick) {
        if (grand_child.mag < Node.MED_MAG-1 &&  child.mag < Node.MED_MAG-1) {
                //System.out.print(" P["+grand_child.Name+"] ");
                return;
                }
        }

        if (grand_child.dist > child.dist) {
                CNode = child.getCNodeAtLink(i);
                if (CNode != null)
                        position_node(child, CNode);
                position_node(child, grand_child);
                position_recursive(child, grand_child);
        }
}
return;
} // end position_recursive


/*
**    METHOD position_node(parent, child) sets the child's x and y vars into
**    screen coordinates based on it's parent's position
**
*/
float position_node (Node parent, Node child) {
double pxSum=0, pySum=0;
Node tn=null;

if (parent == null)
        return(0);
if (child == null)
        return(0);

if (child.Debug && Debug==1) {
        System.out.println("position_node(): parent=<"+parent.Name+"> ->
child=<"+child.Name);
        }
if (parent.Debug && Debug==1) {
        System.out.println("position_node(): parent=<"+parent.Name+"> ->
child=<"+child.Name);
        }

// RUNNING SET OF MAX AND MIN VALUES OF COORDINATES HERE

counter_of_positioned_nodes ++;
```

```
if (min_x > child.x)
        min_x = child.x;
if (min_y > child.y)
        min_y = child.y;
if (max_x < child.x)
        max_x = child.x;
if (max_y < child.y)
        max_y = child.y;


if (child.isLayer) {
        child.x = child.X;
        child.y = child.Y;
        return(0);
        }

if (VRyes(child)) { // draw in absolute position, not relative to caller
        child.x = (0.9*child.X+0.1*child.x); // move 90% each update to ideal
spot
        child.y = (0.9*child.Y+0.1*child.y); // move 90% each update to ideal
spot
   }
else {
    force.calc_pressures(parent,child);
        if (globalMaxPressure > 10) {
                pxSum = 10*force.pxSum/globalMaxPressure;
                pySum = 10*force.pySum/globalMaxPressure;
                }
        else
                {
                pxSum = force.pxSum;
                pySum = force.pySum;
                }
    child.px = pressure_mag*pxSum*Math.abs(pxSum);
    child.py = pressure_mag*pySum*Math.abs(pySum);
    child.x += child.px;
    child.y += child.py;
if (Debug > 0) {
        child.pxParent = force.pxParent;
        child.pxPOV = force.pxPOV;
        child.pxNeighbors = force.pxNeighbors;
        child.pyParent = force.pyParent;
        child.pyPOV = force.pyPOV;
        child.pyNeighbors = force.pyNeighbors;
        }
   }

if (drawEvent) {
if (child.isEvent && parent.isDN) { // force DNs near their event
        parent.x = child.y + 50 + 10/magscale;
```

```
                parent.y = child.y+2 + 2.0*(parent.ChildNum-3);
                parent.ThetaOffset = 0;
                }
        else
        if (parent.isEvent && child.isDN) { // force DNs near their event
                child.x = parent.x + 50 + 10/magscale;
                child.y = parent.y+2 + 2.0*(child.ChildNum-3);
                child.ThetaOffset = 0;
                }
        else
        if (child.isDN && (null != (tn=child.hasSiblingOfType("Event")))) { // else
        force DNs near any related event
                child.x = tn.x + 70 + 10/magscale;
                child.y = tn.y+2 + 1.5*(child.ChildNum-3);
                child.ThetaOffset = 0;
                }
        }

        return(0);
        } // end position_node

        /**
         *   method nsr_render_chromosome
         */
        boolean nsr_render_chromosome (Node node) {
              int x1, y1, x2, y2, t;
              x1 = (int)(node.x - magscale*10) + WindowXcenter;
              y1 = (int)(node.y - magscale*30) + WindowYcenter;
              x2 = (int)(node.x + magscale*10) + WindowXcenter;
              y2 = (int)(node.y + magscale*30) + WindowYcenter;
              graphics.drawLine(x1, y1, x2, y2); // draw a fake chromosome
              graphics.drawLine(x1, y1, x1, y1-(int)(magscale*5));
              t=x1;
              x1=x2;
              x2=t;
              graphics.drawLine(x1, y1, x2, y2); // draw a fake chromosome
              graphics.drawLine(x1, y1, x1, y1-(int)(magscale*5));
              return(true);
              } // end nsr_render_chromosome

        /*
         * trim    return a string of 'length' precision of argument double 'x'
         */
        public String millis (long x) {
        String s;
        int len=0;
                s = java_lang_long.toString(x);
                len = s.length() - 3;
                if (len < 0)
                        len = 0;
        return((String)(s.substring(0,len)));
        } // end trim

 /**
/*
 * prec    return a string of 'length' precision of argument double 'x'
 */
public String prec (double x, int length) {
        String s;
        int max_length, printed_length;;

        s = java_lang_double.toString(x);
        max_length = s.length();
        printed_length = (max_length < length) ? max_length : length;
        return((String)s.substring(0,printed_length));
        } // end prec


/**
 **  state
 **
*/
        public void state () {
        //if (s.equalsIgnoreCase("magscale"))
                P(0,"state","magscale = "+magscale);
        //if (s.equalsIgnoreCase("window"))
                P(0,"state", "WindowXcenter= "+WindowXcenter+" WindowYcenter=
"+WindowYcenter+" WindowXOffset= "+WindowXOffset +"
WindowYOffset="+WindowYOffset);
} // end state



/*************************************************************************
 **
 **  draw_string
 **
*/
        public void draw_string (String s, Point p) {
        draw_string(s, p.x, p.y);
} // end one version of draw_string

/******************************/
        public void draw_string (String s, int x, int y) {
         int i=0, len=50, line_len, start=0, end=0;
        String ss;

// frame.setFont(littleFont);
        end = s.length();
        start = 0;
```

```
if (end <= len)
        graphics.drawString(s, x, y);
else
while (start < end) {
        line_len = end - start;
        if (line_len > len)
                line_len = len;
        if ((s.substring(start+line_len-1, start+line_len).equals(" ")) ||
                (start+line_len == end))
                ss = s.substring(start, start+line_len);
        else
                ss = s.substring(start, start+line_len)+"-";
        graphics.drawString(ss, x, y+(10*i++));
        start += line_len;
        }
// frame.setFont(titleFont);

} // end draw_string
/**********************************************************************/

/**
 *   method show_node_once
 */
void show_node_once (Node node) {
boolean selected;
if (node == null) {
        WARNING(0, "show_node_once", "NULL node");
        return;
        }

// We must switch the graphics context to draw on the live image immediately
//      rather than in double-buffer mode, so use image_counter and handle it.

        graphics = frame.getGraphics();

render_node(null, node, true); // draw immediately, 'true' refers to XOR mode
if (image_counter == 1) { // now undo the damage to the graphics context
'graphics'
                graphics = image1.getGraphics();
                } else {
                graphics = image2.getGraphics();
                }

return;

} // end show_node_once
```

```
 /**
 *   method render_node
 */
float render_node (Node parent, Node child, boolean xor) {
Node tnode = null;
int i, size;
int width=5, height=5;
Point ParentPoint=new Point(), ChildPoint=new Point(), TempPoint=new Point();
int child_center_x, child_center_y;
double max_mag=1;
boolean ZoomOn=false;
int length;  // for the Name, and its formatting
int index;   // for the Name, and its formatting
String Name_text; // for the Name, formatted
String sTS = millis(current_TS - child.TS);
Color current_color=null;
boolean continue_flag=false;



if (child.isEvent && drawEvent)
    continue_flag |= true;
if (child.isCN && drawCN)
    continue_flag |= true;
if (child.isAN && drawAN)
    continue_flag |= true;
if (child.isON && drawON)
    continue_flag |= true;
if (child.isDN && drawDN)
    continue_flag |= true;
if (child.isFile && drawFile)
    continue_flag |= true;
if (child.isURL && drawURL)
    continue_flag |= true;

if (!continue_flag)
       return(0);
/*******************************************************************
*********************************************************************/

if (VRyes(child)) {
       if (child.isEvent) {
               width = (int)(child.size_X*magscale); // in screen coordinates
               height = (int)(child.size_Y*magscale); // in screen coordinates
               }
       else {
               width = (int)(child.mag*child.size_X*magscale); // in screen
coordinates
               height = (int)(child.mag*child.size_Y*magscale); // in screen
coordinates
```

```
                }
        }

// CHILD    ChildPoint is the upper left corner in screen coordinates
 map((int)(child.x), (int)(child.y), ChildPoint);


// TINY CALCS here
if (child == TinyNode) {
        TinyPoint = ChildPoint;
        }

// THIS IS FOR DYNAMIC CONTROL OF HOW MUCH IS SEEN
if (pos_threshold > text_threshold)
        pos_threshold = text_threshold-0.1;
else
        pos_threshold = DEFAULT_POS_THRESHOLD;




// DRAW THING
// DRAW RECTANGLE OR OVAL OR LINE FOR NODE
if (child.mag > box_threshold)
        {
        if (child.isForm) {
                graphics.fill3DRect(ChildPoint.x, ChildPoint.y-height, width,
height, true);
                }
        else if (child.isAN)
                graphics.drawOval(ChildPoint.x, ChildPoint.y-height, width,
height);
        else if (child.isEvent) {
                graphics.fill3DRect(ChildPoint.x, ChildPoint.y-height, width,
height, true);
                }
        else if (child.Type.equals("TL")) {
                graphics.fill3DRect(ChildPoint.x, ChildPoint.y-height, width,
height, true);
                }
        else if (drawBoxes && child.isDN)
                graphics.drawRect(ChildPoint.x, ChildPoint.y-height, width,
height);
        else if (drawBoxes && child.isURL) {
                current_color = graphics.getColor();
                if (child.Data[0]==null) {
                        graphics.setColor(color_obj.Grey);
                        } else {
                        graphics.setColor(color_obj.VeryLightGrey);
                        }
                graphics.fill3DRect(ChildPoint.x, ChildPoint.y-height,
                        (int)(0.7*magscale*width), (int)(magscale*height), true);
                graphics.setColor(current_color);
                }
        if (child.isCN) { //superimpose oval if acting as a CNode
                current_color = graphics.getColor();
                graphics.setColor(color_obj.VeryLightRed);
                graphics.drawOval(ChildPoint.x, ChildPoint.y-height,
                        (int)(2*width), (int)(height));
                graphics.setColor(current_color);
                }
        }
else
        graphics.drawLine(ChildPoint.x, ChildPoint.y-height, ChildPoint.x+4,
ChildPoint.y-height);
if (Details) {
if (
        ((child.x - spaceX) > -5) &&
        ((child.x - spaceX) < 5) &&
        ((child.y - spaceY) > -5) &&
        ((child.y - spaceY) < 5) &&
        (child.mag >= box_threshold)
        )
 ZoomOn = true;
else
 ZoomOn = false;
}
// DRAW NAME
if (((Debug == 1)&&(child.mag>text_threshold)) || (ZoomOn)) {
// Mag, distance
        graphics.drawString("[m(" +prec(child.mag,3)
        +") d(" +child.dist+ ")"
        + child.Name
        +", TS=" +sTS+"]",
        ChildPoint.x, ChildPoint.y+10);
        }
/******************************************************************
        graphics.drawString(child.Name+", Description='"+child.Desc+"'",
ChildPoint.x+width+1, ChildPoint.y);
******************************************************************/
else
if (drawText) {
        if (!drawDN && child.isDN)
        ;
        else {
        if (child.mag > text_threshold) {
                if (child.isURL) {
                        //graphics.setColor(Color.blue);
                        i= 1 + child.Name.lastIndexOf("/"); // add one to work
with substring()
```

```
                        if (i>0) {
                                //base = child.Name.substring(0,i);
                                //name = child.Name.substring(i);
                                draw_string(child.Name.substring(i),
ChildPoint.x+width+1, ChildPoint.y);
                                }
                        }
                else
                        draw_string(child.Name, ChildPoint.x+width+1,
ChildPoint.y);
                }
        }
}

// PARENT
if (parent != null) {

if (child == datasea.Root)
        return(0); // Bail if we are trying to draw the root from another node

  map(parent.x, parent.y, ParentPoint);

// DRAW RELATIONS AS LINES
draw_relations(child, parent, xor);

// In this block, draw relations in yellow from distal ANs
{
        size = child.Links.size();
        for (i=0; i<size; i++) { // check dist of all children, position if
appropriate
                tnode = child.getNodeAtLink(i);
                if (tnode.isAN && tnode != parent)
                        draw_relations(child, tnode, xor);
                }
}
// END DRAW RELATIONS



if ((parent == datasea.Root) || (parent == datasea.POV)) // Special case, else
we won't see it
      graphics.drawString(parent.Name, ParentPoint.x, ParentPoint.y);
  }

// RADIAL LINES FOR LASTNODE
if (child == lastNode)
        {
        if (FlipAxes) {
                child_center_y = (int)(ChildPoint.x + width/2);
                child_center_x = (int)(ChildPoint.y + height/2);
```

```
                        }
                else {
                        child_center_x = (int)(ChildPoint.x + width/2);
                        child_center_y = (int)(ChildPoint.y - height/2);
                        }

        // PRESSURE LINES
        if (Debug > 0) {
                graphics.setColor(Color.blue);
                graphics.drawLine(child_center_x, child_center_y,
                        (int)(child_center_x+10*child.pxPOV), (int)(child_center_y-
        10*child.pyPOV));
                graphics.setColor(Color.red);
                graphics.drawLine(child_center_x, child_center_y,
                        (int)(child_center_x+10*child.pxParent), (int)(child_center_y-
        10*child.pyParent));
                graphics.setColor(Color.green);
                graphics.drawLine(child_center_x, child_center_y,
                        (int)(child_center_x+10*child.pxNeighbors), (int)(child_center_y-
        10*child.pyNeighbors));
                }
        else { // OR MARK WITH AN X
         graphics.drawLine(child_center_x+10, child_center_y+10, child_center_x+30,
        child_center_y+30);
         graphics.drawLine(child_center_x-10, child_center_y+10, child_center_x-30,
        child_center_y+30);
         graphics.drawLine(child_center_x-10, child_center_y-10, child_center_x-30,
        child_center_y-30);
         graphics.drawLine(child_center_x+10, child_center_y-10, child_center_x+30,
        child_center_y-30);
                }
        } // end if child == lastNode
        if (xor)
                graphics.setPaintMode();
        return(0);
        } // end render_node



        /**
         **  draw_relations
         **
        */
        public void draw_relations (Node child, Node parent, boolean xor) {
        int i, size;
        Node tnode;
        Link link=null;
        Color current_color=null;
        Point ParentPoint=new Point(), ChildPoint=new Point(), TempPoint=new Point();
```

```
if (child==null || parent==null)
       return;

map(child.x, child.y, ChildPoint);
map(parent.x, parent.y, ParentPoint);


// DRAW RELATIONS AS LINES
//if ((parent.isCN || child.isCN) && !drawCN)
//      ;
if ((parent.isAN || child.isAN) && !drawAN)
        ;
else if ((parent.isDN || child.isDN) && !drawDN)
        ;
else if (mode_obj.lines_mode.equals("none"))
        ;
else if (child == datasea.Root)
        ;
else if ((child.mag >= relations_threshold) && (parent.mag >=
relations_threshold)) {
if (xor)
       graphics.setXORMode(Color.green);
else
       color_obj.set_color_for_relations(graphics, parent, child);

if (mode_obj.lines_mode.equals("local")) {
    if (lastNode != null) {
    if ((lastNode == parent)||(lastNode == child)) {
        //     graphics.drawLine(ChildPoint.x, ChildPoint.y,
        //                   ParentPoint.x, ParentPoint.y);
        size = child.Links.size();
        for (i=0; i<size; i++) { // check dist of all children, position if
appropriate
               tnode = child.getNodeAtLink(i);
               map(tnode.x, tnode.y, TempPoint);
               graphics.drawLine(TempPoint.x, TempPoint.y,
                            ChildPoint.x, ChildPoint.y); // child to its
siblings

        }
    }
   }
}

else if (mode_obj.lines_mode.equals("all")) {
        if (child.isSelected == true) // special case for traceback
            graphics.setColor(Color.red);

       boolean is_aliased_time = false;
```

```
        if (child==datasea.pop.yesterday_node || child==datasea.pop.today_node
|| child==datasea.pop.tomorrow_node
                || child==datasea.pop.last_week_node ||
child==datasea.pop.this_week_node || child==datasea.pop.next_week_node)
                        is_aliased_time = true;

        if (child.isEvent && parent.isEvent && !is_aliased_time)
            graphics.drawLine(ChildPoint.x, ChildPoint.y,
                            ParentPoint.x, ChildPoint.y); // Event to TimeLine
        else
        if (parent != datasea.POV)
                graphics.drawLine(ChildPoint.x, ChildPoint.y,
                            ParentPoint.x, ParentPoint.y); // child to parent
        }

if (drawLinkNames
        && ((parent.mag > Node.BIG_MAG && child.mag > Node.BIG_MAG)
                || (parent==gui.lastNode || child==gui.lastNode)))   {
int x1=0, y1=0, x2=0, y2=0;

link = parent.getLinkTo(child);
if (link != null) {
if (link.Name != "")
graphics.drawString("("+link.Name+")",
        (int)((ParentPoint.x+ChildPoint.x)/2),
        (int)((ParentPoint.y+ChildPoint.y)/2));
if (parent.getPol(child) == '+') {
        x1 = (int)ParentPoint.x;
        y1 = (int)ParentPoint.y;
        x2 = (int)(ParentPoint.x+(ChildPoint.x-ParentPoint.x)*0.7);
        y2 = (int)(ParentPoint.y+(ChildPoint.y-ParentPoint.y)*0.7);
        graphics.drawOval( x2, y2, 3, 3);
        //graphics.drawLine( x1, y1, x2, y2);
                // child to parent
        }
else
if (parent.getPol(child) == '-') {
        x1 = (int)ParentPoint.x;
        y1 = (int)ParentPoint.y;
        x2 = (int)(ParentPoint.x+(ChildPoint.x-ParentPoint.x)*0.3);
        y2 = (int)(ParentPoint.y+(ChildPoint.y-ParentPoint.y)*0.3);

        graphics.drawOval( x2, y2, 3, 3);
        //graphics.drawLine( x1, y1, x2, y2);
        }
else {
        x1 = (int)ParentPoint.x;
        y1 = (int)ParentPoint.y;
        x2 = (int)(ParentPoint.x+(ChildPoint.x-ParentPoint.x)*0.7);
        y2 = (int)(ParentPoint.y+(ChildPoint.y-ParentPoint.y)*0.7);
        graphics.drawOval( x2, y2, 5, 3);
        //graphics.drawLine( x1, y1, x2, y2);
        x2 = (int)(ParentPoint.x+(ChildPoint.x-ParentPoint.x)*0.3);
        y2 = (int)(ParentPoint.y+(ChildPoint.y-ParentPoint.y)*0.3);
        graphics.drawOval( x2, y2, 5, 3);
        //graphics.drawLine( x1, y1, x2, y2);
        }
}
}


// Special case drawing of relations, if isSelected
if (child.isSelected) {
current_color = graphics.getColor();
        size = child.Links.size();
        for (i=0; i<size; i++) { // check dist of all children, position if
appropriate
                tnode = child.getNodeAtLink(i);
                map(tnode.x, tnode.y, TempPoint);
// set the colors
                if (tnode.dist == child.dist)
                        graphics.setColor(Color.green);
                else
                if (tnode.dist >= child.dist+0.1)
                        graphics.setColor(Color.blue);
                else
                if (tnode.dist == child.dist-1)
                        graphics.setColor(Color.red);
                else
                        graphics.setColor(Color.yellow); // not within dist of 1
                graphics.drawLine(TempPoint.x, TempPoint.y,
                        ChildPoint.x, ChildPoint.y); // child to its
siblings
        }
graphics.setColor(current_color);
}

} // mag thresholds

} // end draw_relations


/**
 *   method createMyself    Creates a node, links it to 'this'
 */
        public void createMyself () {
        if (GlobalNodeNode == 0) {
                GlobalNodeNode ++;
                Myself = new Node("Node", "Java Object", "The Essential Node");
                P(0, "GUI.createMyself", "Initializing the Myself Object");
                Myself.describe("To Console");
                }
     } // end createMyself


    /**
     ** demo1
     */
    public  void demo1 () {
        int i;
        Node a, b;

        P(0,"demo1", "Begun");
    input.string_input("show thes");
    input.string_input("3");
    input.string_input("potmag egg art");
    input.string_input("potmag egg art");
    input.string_input("potmag egg art");

        P(0,"demo1", "Done.");

    } // end demo1

    /**
     ** demo2
    */
    public  void demo2 () {
            int i;

           }  // end demo2

    /**
     ** demo2
    */
    public  void demo3 () {
           } // end demo3

    /**
     ** method run_thread
    */
           public void run_thread () {
           Thread active_thread;

    active_thread = Thread.currentThread();

    if (animation_thread != null) {
           WARNING(0,"run_thread","Starting thread, old one exists.  "+
                   "(animation_thread = "+animation_thread+")");
           animation_thread.start();     // should invoke run() below
           button_animate.setLabel("Running");
        return;
        }

debug_animation_thread(1,"run_thread", "animation thread is null, creating new
Thread(this)");

animation_thread = new Thread(this);
animation_thread.start();     // should invoke run() below

button_animate.setLabel("Running");

debug_animation_thread(1,"run_thread", "after new Thread(this)");

P(0, "run_thread", "Thread started: "+animation_thread);
} // end run_thread


/**
 **  debug_animation_thread
 **
*/
        static public void debug_animation_thread (int debug_level, String
fn_name, String fn_desc) {

if (animation_thread == null)
        P(debug_level, "debug_animation_thread["+fn_name+"]: ",fn_desc+":<null,
activeCount="+Thread.activeCount()+"("+Thread.currentThread()+")>");
else {
        P(debug_level, "debug_animation_thread["+fn_name+"]:
",fn_desc+":<isAlive="+animation_thread.isAlive()+",
activeCount="+Thread.activeCount()+"("+Thread.currentThread()+")>");
}


} // end debug_animation_thread



/**
 ** method stop_thread
 **           This should only be called from a point guaranteed not to be
 **           inside a routine recursively following the node network, e.g.
 **           from update() which checks StopThreadRequest to see if we
 **           should be called.
*/
        public void stop_thread () {
        Thread active_thread;

        P(0, "stop_thread","stopping animation_thread");
```

```
if (animation_thread != null) {
               if (animation_thread.isAlive()) {
               P(0,"stop_thread", "isAlive(), calling animation_thread.stop(),
then nulling it");
               animation_thread.stop();
               animation_thread = null;
               }
               else
                      debug_animation_thread(1,"stop_thread", "not alive, not
stopping it...");

/****************************************************************
//             try { animation_thread.join(1000); }
//                    catch (InterruptedException i) {
//                    ERROR(0,"stop_thread","ERROR FROM 'join(1000)',
InterruptedException "+i);
//             }
//             animation_thread.stop();
****************************************************************/
}
       else
               P(0, "stop_thread","animation_thread  is null");
debug_animation_thread(1,"stop_thread", "apparently animation_thread is null");

button_animate.setLabel("Stopped");
//datasea.needdistUpdate = true;
       } // end stop_thread


/**
 ** start
 **
*/
       public void start () { // Applet is started, don't do anything special
               P(0, "start","ran.");
       }
       public void stop () {  // Applet is stopped, stop thread also
       if (animation_thread != null) {
               if (animation_thread.isAlive()) {
               P(0,"stop", "!=null, isAlive so running stop_thread()");
               stop_thread();
               }
       }
       else {
               P(0, "stop","ran. 'animation_thread' = null");
       }
       } // end start

/**
```

```
 ** sleep
 **
*/
        public static void sleep (int millis) {
        try { Thread.sleep(millis); }
                catch (InterruptedException e) { ; }
} // end sleep

/**
 ** run
 **
*/
        public void run () {   // This is for the thread animation_thread
                int i;
                date = new java.util.Date();
                timestamp = new Timestamp(date.getTime());

System.err.println(" --- run(), BEGUN --- ");
        while (true) {
                sleep(10);
                update(true);
                if (StopThreadRequest) {
                        Animating = false;
                        System.err.println(" --- run(), StopThreadRequest --- ");
                        break; // thread should be done by leaving this function
                        }
                }
        }

/**
 ** i    convert to integer
 **
*/
        public static int i (float x) {
                return ( (int)x );
                }
        public static int i (double x) {
                return ( (int)x );
                }


/**
 **  status
 **
*/
static public void status (String string) {
int i;

for (i=14; i>0; i--)
        StatusLine[i] = StatusLine[i-1];
```

```
StatusLine[0] = string;

} // end status


/**
 ** WARNING       Print WARNING messages
 **
*/
public  static void WARNING (int debug_thresh, String fn, String string) {
        if (debug_thresh <= Debug) {
                if (list != null)
                list.add("==> WARNING in "+fn+"(): ======= '"+string+"'", 0);
                System.err.println("==> WARNING in "+fn+"():    "+string);
                // debug_animation_thread(1, fn, string);
        status("WARNING: "+string);
         }
} // end WARNING


/**
 ** ERROR       Print ERROR messages
 **
*/
public  static void ERROR (int debug_thresh, String fn, String string) {
if (debug_thresh <= Debug) {
        beep();
        if (list != null)
        list.add(" ============ ERROR in function "+fn+"(): =======
'"+string+"'", 0);
        System.err.println(" ============ ERROR in function "+fn+"():
"+string);
        dump_stack(fn);
//      global_str_size = 1;
//      global_str[0] = "ERROR in fn "+fn+"():"+string;
//      sleep(3000);// Erase this after a bit ...
//      global_str_size = 1;
//      global_str[0] = "(prior error was in fn "+fn+"():"+string+")";
        status("ERROR: "+string);
         }
}



/**
 ** P       Print debug messages   if given value == current Debug value
 **
*/
public  static void P (int debug_thresh, String fn, String string) {
if (debug_thresh == Debug) {
        if (list != null) {
               list.add("  "+fn+"():    "+string, 0);
               System.err.println("  "+fn+"():    "+string);
               }
               //status(string);
        }
}



/**
 **  dump_stack
 **
*/
        static public void dump_stack (String fn_name) {
        System.err.println("dump_stack(): Dumping currentThread's stack trace
deliberately, by "+fn_name);
        System.err.println("dump_stack():
==============================================");
        (Thread.currentThread()).dumpStack();
        System.err.println("dump_stack():
==============================================");
} // end dump_stack

 /**
 ** method print_tree
*/
public void print_tree (String[] words, int num_words) {
Node node;

if (num_words==1)
       node = gui.lastNode;
else
       node = datasea.find_node_named(words[1]);

if (node != null) {
       print_tree_data_start(node);
       } // end print_tree
else
        GUI.WARNING(0,"print_tree", "No node given");
} // end print_tree



/**
 **  clear_global_str
 **
*/
public void clear_global_str () {
       global_str_size = 0;
```

```
} // end clear_global_str

/**
/**
 **  add_to_global_str
 **
*/
public void add_to_global_str (String str) {
if (global_str_size == (MAX_GLOBAL_STR_SIZE-2)) {
        WARNING(0,"add_to_global_str","Over-run in global_str_size");
        global_str[global_str_size++] = "------out of room in global_str---------
----------";
        return;
        }
if (global_str_size > (MAX_GLOBAL_STR_SIZE-2)) {
        WARNING(0,"add_to_global_str","Over-run in global_str_size");
        return;
        }

P(0,"add_to_global_str", str);
global_str[global_str_size++] = str;
return;
} // end add_to_global_str

/**
 **  print_tree_data_start
 **
*/
public  void print_tree_data_start (Node node) {
        int i, j, size, lines_count=1;
        Node tn;

        clear_global_str();

        size = node.Links.size();
        add_to_global_str("-----------------------------------------------------
---");

        add_to_global_str(node.Name+"("+node.Type+", "+node.Desc+"):
"+node.Data);
        for (i=0; i<size; i++) {
                tn = node.getNodeAtLink(i);
                //if (tn.dist == node.dist+1)
                if (tn.dist > node.dist)
                        lines_count += print_tree_data_recursive(tn);
        }

        add_to_global_str("-----------------------------------------------------
---");
        add_to_global_str("Lines returned = "
                    +lines_count+", global_str_size="+global_str_size);
} // end print_tree_data_start

/**
 **  print_tree_data_recursive
 **
*/
 public  int print_tree_data_recursive (Node node) {
        int i, j, size, spaces_count, line_count=1;
        Node tn;
        String spaces;
        String s=null;


        spaces_count = 60-(int)(node.mag*10.0);
        .spaces = " ";
        for (j=0; j<spaces_count; j++) // concatenate spaces_count " "
                spaces = spaces + " ";
        if (node.mag > 1)
                add_to_global_str(spaces+node.Name+"("+node.Type+",
mag="+prec(node.mag,3)+")");
        size = node.Links.size();
        for (i=0; i<size; i++) {
                tn = node.getNodeAtLink(i);
                //if (tn.dist == node.dist+1)
                if (tn.dist > node.dist)
                        line_count += print_tree_data_recursive(tn);
        }
return(line_count);
    } // end print_tree_data_recursive

/**
 **  dump_data
 **
*/
        public  void dump_data (int debug_value, Node node) {
                int i, j, size;
         P(debug_value,"dump_data",node.Name+" Desc=("+node.Desc+"),
Data=:"+node.Data);
    }

/**
 **  dump_node
 **
*/
        public  void dump_node (int debug_value, boolean show_links, Node node) {
                int i, j, size;
                String str;
                 Node tnode;
                Link tlink=null;
        if (Debug < debug_value)
                return;

        if (node == null) {
                ERROR(0, "dump_node", "node is null");
                return;
                }
        size = node.Links.size();

        for (j=0; j<recursion_depth; j++) {

                System.err.println("------------------------------------------
----------------------------------------");
                str = "dmp:'" +node.Name
                        + ",dist=" +node.dist
                        + ",Tdist=" +node.Tdist
                        +",mag="+ prec(node.mag,3)
                        + "'  '" +node.Type+ "'"
                        + "' isEvent=" +node.isEvent+ ","
                        + "' isDN=" +node.isDN+ ","
                        + "' isAN=" +node.isAN+ ","
                        + "' isFile=" +node.isFile+ ","
                        + "' isDirectory=" +node.isDirectory+ ","
                        + "' isCN=" +node.isCN+ ","
                        + "' isURL=" +node.isURL+ ","
                 + ",ChildNum=" +node.ChildNum
                 + ",BigChildCount=" +node.BigChildCount
                        + " (x,y)=(" +i(node.x)+ "," +i(node.y)+")"
                        + " (size_x,size_y)=(" +i(node.size_x)+ ","
+i(node.size_y)+")"
                        + " (X,Y)=(" +i(node.X)+ "," +i(node.Y)+")"
                        + " (size_X,size_Y)=(" +i(node.size_X)+ ","
+i(node.size_Y)+")" ;
                list.add(str, 0);
                System.err.println(str);
                }
        if (show_links)
                {
                for (i=0; i<size; i++)
                        {
                        tnode = node.getNodeAtLink(i);
if (tnode == node)
        System.out.println("dump: node=child, <"+node.Name+">");
                        str = "      "
                        +tnode.Name +"     "
                          +",dist="+ tnode.dist
                        + ",Tdist=" +node.Tdist
                          +" ,mag="+ prec(tnode.mag,3)
                        + "'  (" + tnode.Type+ ")"
                        + " CS("+prec(node.get_CS(tnode),3)+")"
                    + ",ChildNum=" +tnode.ChildNum
                    + ",BigChildCount=" +node.BigChildCount;
                         list.add(str, 0);
                        System.err.println(str);
                         }
                   }
          if (node.isCN)
                   {
                   size = node.ContextLinks.size();
                   for (i=0; i<size; i++) {
                   tlink = (Link)(node.ContextLinks.elementAt(i));
                        if (tlink != null) {
                        str =  "   dmp:ContextLink[" +i+"]"
                        +"' links nodes `" +tlink.NodeR.Name+ "' and
'"+tlink.NodeL.Name+"'";
                        System.err.println(str);
                         }
                   }
          }
          System.err.println("-----------------------------------------------------
----------------------------------");
          return;
          } // end dump_node()

/**
 **  dump_nodes
 **
*/
          public  void dump_nodes (int debug_value, Node node) {
                   int i, j, size;
           if (node == null)
               return;
               dump_nodes_recursive(debug_value, node);
          } // end dump_nodes()

/**
 **  dump_nodes
 **
*/
          public  void dump_nodes_recursive (int debug_value, Node node) {
                   int i, j, size;
                   if (recursion_depth == 0)
                   P(debug_value, "dump_nodes", "---------------------------");
                   dump_node(debug_value, false, node);
                   recursion_depth ++;
           size = node.Links.size();
           for (i=0; i<size; i++) {
               if (node.getNodeAtLink(i).hasLargerDistThan(node))
                    dump_nodes(debug_value, node.getNodeAtLink(i));
```

```
                }
                   recursion_depth --;
                   if (recursion_depth == 0)
                   P(debug_value, "dump_nodes", "---------------------------");
                   return;
          } // end dump_nodes_recursive()

/**
 **  run1
 **
*/
public void run1 () {

        update(1);

} // end run1


/**
 ** method update
 ** paint nodes, starting on Root
*/
    public void update (int count) {
        int i;
        for (i=0; i<count; i++)
            update(true);
    } // end update(int)


/**
 **  request_stop_thread   block and wait for a limited while until Animating is
false, return ultimately.
 **
*/
static public void request_stop_thread () {
int i;

StopThreadRequest = true; // sensed by ...

for (i=0; (i<100 && Animating==true); i++) {
        System.err.print(".");
        sleep(10);
        }
return;
} // end request_stop_thread

/**
 ** method update
 ** update and paint nodes, starting on Root
*/
```

```
        public void update (boolean change_images) {

// change image_counter and get appropriate graphics in prep for paint()

if (change_images) {
        if (image_counter>=2)
                {
                image_counter=1;
                graphics = image1.getGraphics();
                }
        else
                {
                image_counter=2;
                graphics = image2.getGraphics();
                }
        }

// SET THE GRAPHICS TO THE IMAGE
         if (datasea.POV != null) {
                paint(datasea.POV, 0);
            }
            else {
                paint(datasea.Root, 0);
            }
graphics.setColor(color_obj.LightGrey);

// SET GRAPHICS TO THE FRAME'S GRAPHICS IN PREP' FOR drawImage()
graphics = frame.getGraphics();
this.getToolkit().sync();
// DRAW THE IMAGE ONTO THE FRAME
if (image_counter==1)
        graphics.drawImage(image1, 0,0, frame);
else
        graphics.drawImage(image2, 0,0, frame);

if (auto_rescale)
        rescale("run");
} // end update

/**
 ** method reset_current_TS , called by update
*/
public static void reset_current_TS () {
     current_TS = java.lang.System.currentTimeMillis();
} // end reset_current_TS


/**
 **  set_Xnode
 **
```

```
*/
public void set_Xnode () {
int i, size;
Node child;

if (lastNode == Xnode) {
        Xnode = null;
        P(0,"set_Xnode", "Setting Xnode to null");
        }
else
if (lastNode != null) {
        Xnode = lastNode;
        P(0,"set_Xnode", "Setting Xnode to lastNode "+Xnode.Name);
        }
else
        WARNING(0,"set_Xnode", "Need a lastNode to set Xnode.");


} // end set_Xnode


/**
 ** method paint , called by update
*/
public void paint (Node POV, int dist_to_POV) {
       float completion_level = 0;
        reset_current_TS();
        if (Debug==1)
                timer.start_timer("position");
rescale("clear");
if (Xnode != null) { // 11/27/99
        DataSea.needdistUpdate = true;
}
completion_level  =   position_start(POV);

if (Xnode != null) { // 11/27/99
        DataSea.needdistUpdate = true;
        completion_level  =   position_start(Xnode);
        }

if (Debug==1)
        timer.end_timer("position");
if (Debug==1)
        timer.start_timer("render");

completion_level  =   render_start(POV);

if (Debug==1)
        timer.end_timer("render");
```

```
} // end paint


  /**
   ** get_angle
   ** set the angle theta of a node to angle from it's caller
   **
   */
   static public double get_angle(Node a, Node b) {
   double delta_x, delta_y;
   delta_x = b.x - a.x;
   delta_y = b.y - a.y;
   return (get_angle(delta_x, delta_y));
   } // End of get_angle (Node...)

  /**
   ** get_angle
   ** set the angle theta of a node to angle from it's caller
   **
   */
   static public double get_angle(double delta_x, double delta_y) {
   return(Math.atan2(delta_y,delta_x));
   } // End of get_angle (double...)



/**
 **  action
 **
*/
public void action (Node node) {
int i, size, j, sizej;
Node child, grand_child;


if (node == null)
        return;

if (node.isURL) {
        text_frame.setVisible(true);
        text_graphics = text_frame.getGraphics();
        text_frame.setTitle(node.Name);
        if (node.Data[0] != null) {
                text_graphics.clearRect(0,0, (text_frame.getSize()).width,
(text_frame.getSize()).height);
                draw_text(node);
                }
        }
else
if (node.isAN) {
        text_frame.setTitle("(Hidden)");
        text_frame.setVisible(false);
        }
else
if (node.isDN) {  // something like a menu
        text_frame.setTitle("URL not selected");
        text_frame.setVisible(false);

        node.set_mag(Node.MAX_MAG);
        datasea.back_r((Node)null, node, 0);
        datasea.back_r((Node)null, node, 0);
        datasea.back_r((Node)null, node, 0);
        size = node.Links.size();

// make children all visible
        for (i=0; i<size; i++) {
                child = (Node)(node.getNodeAtLink(i));
                if (child.dist > node.dist) {
                        child.set_mag(Node.MAX_MAG - 1);
                        sizej = child.Links.size();

// make grand-children's lines all visible, but no names
                        for (j=0; j<sizej; j++) {
                                grand_child = (Node)(child.getNodeAtLink(j));
                                if (grand_child.dist > child.dist) {
                                        grand_child.set_mag(text_threshold - 0.1);
                                        }
                                }
                        }
                }
        }

datasea.needdistUpdate = true;

} // end action


/**
 **  draw_text
 **
*/
public void draw_text (Node node) {
int i, size;
String s=null;

// TEST HERE

size = Node.MAX_TEXT_DATA_LINES;
for (i=0; i<size; i++) {
        s = node.Data[i];
        if (s == null)
                break;
        else {
                s = eliminate_html(s);
                        text_graphics.drawString(s, 10,50+10*i);
                }
        }

} // end draw_text



/**
 **  find_domain_name
 **
*/
public String find_domain_name (String input_string) {
int index, num_words;
String words[], ret_string=null;


if (-1 == (index = input_string.toLowerCase().indexOf("http://")))
        return((String)null);

System.out.println("find_domain_name: full string is "+input_string);

StringTokenizer st = new StringTokenizer(input_string, "/" );
num_words = st.countTokens();
words = new String[num_words];

if (num_words >= 2) {
        ret_string = st.nextToken();
System.out.println("find_domain_name: 'http:' is ================ "+ret_string);
        ret_string = st.nextToken(); // use the second one
System.out.println("find_domain_name: Assuming domain is ================
"+ret_string);
        }

/*******************************************************
for (int i = 0; i < num_words; i++) {
      - words[i] = st.nextToken();
        index = words[i].toLowerCase().indexOf("www.");
        if (0 <= index)
                ret_string = words[i];
        }
*******************************************************/
return(ret_string);
} // end find_domain_name

/**
```

```
 **  find_URL_name
 **
*/
public String find_URL_name (String input_string) {
int index, num_words;
String ret_string=null;
String words[];

if (-1 == (index = input_string.toLowerCase().indexOf("href=")))
        return((String)null);

StringTokenizer st = new StringTokenizer(input_string, " ,=<>\"\t\r\n" );
num_words = st.countTokens();
words = new String[num_words];
//System.out.println("find_URL_name: full string is -->"+input_string+"<--,
num_words="+num_words);

for (int i = 0; i < num_words; i++) {
        words[i] = st.nextToken();
        //System.out.println("find_URL_name: Working on ================
"+words[i]);
        index = words[i].toLowerCase().indexOf("href");
        if (0 <= index) { // found one
                if ((i+1)<num_words) {
                words[++i] = st.nextToken();
                ret_string = words[i];
                }
                else
                System.out.println("find_URL_name ERROR, FOUND 'HREF' but i+1 >=
num_words");
        }
}

// See if there are bad things here ...
if (ret_string != null) {
if (0 <= (index = ret_string.toLowerCase().indexOf("mailto")))
        ret_string = null;
else
if (0 <= (index = ret_string.toLowerCase().indexOf("cgi-bin")))
        ret_string = null;
else
if (0 <= (index = ret_string.toLowerCase().indexOf("#")))
        ret_string = null;
else
if (0 <= (index = ret_string.toLowerCase().indexOf("messages"))) // for egg-
tempera
        ret_string = null;
        }
//if (ret_string != null)
//      System.out.println("find_URL_name returning "+ret_string);
```

```
return(ret_string);

} // end find_URL_name


/**
 **  eliminate_html   eliminate what look like HTML tags, replace with a blank
 **
*/
public String eliminate_html (String string_arg) {
String s=string_arg;
int left_bracket_index, right_bracket_index;
String w1, w2;

        while (true) {
        if (s.length() <= 0) {
                break;
                }
        if (0 <= s.toLowerCase().indexOf("meta")) {
                break;
                }
        left_bracket_index = s.indexOf("<");
        right_bracket_index = s.indexOf(">");
        if (left_bracket_index < 0)
                break;
        if (right_bracket_index < 0)
                break;
        if (right_bracket_index <= left_bracket_index) {
                //System.err.println("draw_text "+s);
                //System.err.println("draw_text breaking:right_bracket_index <=
left_bracket_index");
                break;
                }
        w1 = s.substring(0, left_bracket_index);
        w2 = s.substring(right_bracket_index+1);
        s = w1+" "+w2;
        }
return(s);
} // end eliminate_html


/**
 **  eliminate_punctuation   eliminate what look like HTML tags, replace with a
blank
 **
*/
public String eliminate_punctuation (String string_arg) {
String s=string_arg;
int left_bracket_index, right_bracket_index;
```

```
String w1, w2;

        while (true) {
        if (s.length() <= 0) {
                break;
                }
        if (0 <= s.toLowerCase().indexOf("meta")) {
                break;
                }
        left_bracket_index = s.indexOf(",");
        right_bracket_index = s.indexOf(">");
        if (left_bracket_index < 0)
                break;
        if (right_bracket_index < 0)
                break;
        if (right_bracket_index <= left_bracket_index) {
                //System.err.println("draw_text "+s);
                //System.err.println("draw_text breaking:right_bracket_index <=
left_bracket_index");
                break;
                }
        w1 = s.substring(0, left_bracket_index);
        w2 = s.substring(right_bracket_index+1);
        s = w1+" "+w2;
        }
return(s);
} // end eliminate_punctuation


/**
 **  p    print out something
 **
*/
public void p (String s) {
System.out.println(s);
} // end p

/**
 **  node_to_string
 **
*/
public String node_to_string (Node node) {

String s =
"<"+node.Name+">(m="+node.mag+")("+node.Type+")[d="+node.dist+"][Td="+node.Tdist
+"]";
return(s);
} // end node_to_string
} // End of Object GUI
```

```
// This is DataSea.java     by Rocky Nevin

import java.applet.*;
import java.lang.*;
import java.awt.*;
import java.util.*;
import java.io.*;



/**
 * This is DataSea.java     by Rocky Nevin
 * @version     0.1, 8/6/98
 * @version     0.4, 3/12/98
 *
 * 8/6/98
 * New version, rewritten
 * Ported to JBuilder2 9/98

 * Principal Methods:
   absorb_POV
   add_POV
   add_to_node_vec
   add_to_unprocessed_vec
   cloner
   connect_up
   find_node*
   gen_*
   init
   locate_node
   more_attraction/repulsion
   populate
   PostProcessor
   show_node_vec
   spread_mag
   string_input
   word_*  methods parsing input like 'show', 'back', 'more', 'sim' ...
           These all used to be word_xxx or word_xxx_start, but many now are
           simply xxx, like 'word_back_start' is now 'start'.

1/19/99
Node-specific rendering is important and must be included. Nodes with
master-rendering function will invoke their children's subservient rendering.
1/20/99
Add potentiate function to the spread mechanism: potentiate 1+ nodes
which spreads, and then stimulate 1+ other nodes. Any nodes both
stimulated and potentiated change their magnitude, others that are only
stimuated are not changed, or at least not as much.
4/15/99
Add Group(int level) which will go from POV to dist=level and from
```

```
there force all children of dist=(level-1) to position themselves
on parent (dist=level), and travel back up proximally to POV. Subsequent
invocations on (level-1) will group things reasonably.
*/


public class DataSea extends Object {


static GUI gui; // The GUI passed to the DataSea constructor, which created us
static Node currentCNode=null; // to link POV to easily
static Node CN; // to link POV to easily
static Node Notes; // to link POV to easily
static Node Myself; // This is a self-node, one which can be used to show the
DataSea
                    // program itself
static Node Root;
static String useless_words; // exclude these ... make exclusion list. discard
static Node Thesaurus_node;
static Populate pop;
// static Node BG;
static Node POV = null;
static Vector node_vec; // holds all nodes for searching later
static Vector list_vec; // holds nodes for passing between routines
static Vector big_mag_node_vec; // holds nodes for passing between routines
static int GlobalDistalCount = 0;
static int event_counter = 0;
Vector GlobalVec = null;
Vector unprocessed_vec;
// int maxnodes=1000;
static int link_ID = 0;
int POV_namer = 0;
static boolean needdistUpdate=false;
static boolean theta_org=true;
static double delta_dist = 0.1; // used to increment Node.dist when types are
same
boolean whats = false;



public DataSea (Object gui_argument) {        // Constructor
gui = (GUI)gui_argument;    // Now we can refer to our creator
this.init();
} // end DataSea constructor

public void init () {
       pop = new Populate(this);
       init_useless_words();
if (Myself == null) {
       }
return;
}


/**
 **  init_useless_words
 **
*/
public void init_useless_words () {
int i, size;
Node child;

String w;
w = "a an any are as at be by fix have has html if is rel see the to me not on
pdt public ";
w = w+"this written about how's going would be it";
w = w+" said says with you don't forget close please your";
w = w+" for next turn dear named my day happy input";
w = w+" interesting believes of most";
w = w+" yesterday today tomorrow lastweek thisweek";
w = w+" -- < > = & [ ] \\ / . ; : .";
w = w+" Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec";
w = w+" 1999 2000";
w = w+" 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15";
w = w+" 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31";

// Ignore case
useless_words = w.toLowerCase();

//System.out.println("Useless_words: "+useless_words);
return;

} // end init_useless_words


/**
 **  reset_node_IDs
 **
*/
       public void reset_node_IDs () {
        int size, i;
        Node node, child;

size = node_vec.size();
for (i=0; i<size; i++) {
       node = (Node)(node_vec.elementAt(i));
       node.ID = 0;
       }
} // end reset_node_IDs
```

```
/**
 **  cats   Lift mag of downstream nodes of same type.
 **                  If different type, lift half and quit
*/
public  void cats (String words[], int num_words) {
int i, size;
Node node, child;

if (null == (node = figure_out_node("cats", words, num_words)))
        return;

reset_node_IDs();

node.lift(2); // make sure the starting point isn't left in the dust

size = node.Links.size();
for (i=0; i<size; i++) {
        child = node.getNodeAtLink(i);
        if (child.hasLargerDistThan(node)) {
                System.out.println("cats "+child.Name);
                cats(node, child, i+1);
                }
        }

} // end cats

/**
 **  cats   Show categories (ANs) from node by calling new_back_r on ANs hit >
once
 **
*/
public  void cats (Node caller, Node node, int id) {
int i, size;
Node child;

if (node==null)
        return;

if (node.ID != 0)
        node.ID = id;
else {
        back_r(caller, node, -10);
        //new_back_r(caller, node, -10, -10);
        return;
        }

size = node.Links.size();
for (i=0; i<size; i++) {
		child = node.getNodeAtLink(i);
		if (child.hasLargerDistThan(node)) {
			System.out.println("cats "+child.Name);
			cats(node, child, id);
			}
		else {
			System.out.println("cats halting on "+child.Name);
			}
		}

	} // end cats



	/**
	 **  magdownstream   Lift mag of downstream nodes of same type.
	 **                  If different type, lift half and quit
	*/
	public  void magdownstream (String words[], int num_words) {
	int i, size;
	Node node, child;

	if (null == (node = figure_out_node("magdownstream", words, num_words)))
		return;

	node.lift(2); // make sure the starting point isn't left in the dust

	size = node.Links.size();
	for (i=0; i<size; i++) {
		child = node.getNodeAtLink(i);
		if (child.hasLargerDistThan(node))
		if (node.goesDownstreamTo(child) && node.Type==child.Type) {
			child.lift(2);
			System.out.println("magdownstream "+child.Name);
			magdownstream(child);
			}
		else {
			child.lift(2);
			System.out.println("magdownstream halting on "+child.Name);
			}
		}

	} // end magdownstream

	/**
	 **  magdownstream
	 **
	*/
	public  void magdownstream (Node node) {
	int i, size;
```

```
Node child;

if (node==null)
        return;



size = node.Links.size();
for (i=0; i<size; i++) {
        child = node.getNodeAtLink(i);
        if (child.hasLargerDistThan(node))
        if (node.goesDownstreamTo(child) && node.Type==child.Type) {
                child.lift(2);
                System.out.println("magdownstream "+child.Name);
                magdownstream(child);
                }
        else {
                child.lift(1);
                System.out.println("magdownstream halting on "+child.Name);
                }
        }

} // end magdownstream




/**
 ** method backs      amplify mag going backwards, spread to first children also.
Calls new_back_r
*/
        public void backs (String[] words, int num_words) {
        int i, size;
        Node node, tnode;

        if (num_words<1)
                return;

// HANDLE IDENTIFYING THE CORRECT NODE TO START ON
        if (num_words==1)
                node = gui.lastNode;
        else
                node = find_node_named(words[1]);

        if (words[0].equalsIgnoreCase("whats")) {
                whats = true;
                }

// CHECK FOR ERRORS
        if (node==null) {
```

```
        if (num_words>1)
            GUI.WARNING(0,"DataSea.backs","Can't find node "+words[1]);
        else
            GUI.WARNING(0,"DataSea.backs","Neither Name given nor existing
lastNode.");
                return;
                }
        else
                GUI.P(1,"DataSea.backs","Found node named '"+node.Name+"'");
backs(node);
}




/**
 ** method backs     amplify mag going backwards, spread to first children also.
Calls new_back_r
*/
        public void backs (Node node) {
        int i, size;

boolean Saved_StopSpread = node.StopSpread; // Temporarily change it
node.StopSpread = true;

GUI.SavedNode = node; // used by 'why' later

//System.out.println("backs(): new_back_r("+node.Name+")");
//new_back_r((Node)null, node, 0, false);

Node tn=null;
// ------ for children --------------------
size = node.Links.size();
for (i=0; i<size; i++) {
        tn = (Node)(node.getNodeAtLink(i));
        if (tn.dist != -1) { // That is, there is a path to POV from here ...
                //System.out.println("");
                //System.out.println("backs(): new_back_r("+tn.Name+")");
                new_back_r((Node)node, tn, 0, false);
                }
}
// ----- end for children -----------------

node.StopSpread = Saved_StopSpread; // Restore it

return;
} // end backs
```

```
/**
 ** method back     amplify mag going backwards, calls back_r
*/
        public void back (String[] words, int num_words) {
        int i, size;
         Node node, tnode;

if (null == (node = figure_out_node("back", words, num_words)))
        return;

        if (words[0].equalsIgnoreCase("whats")) {
               whats = true;
               }
// NOW, MAKE RECURSIVE CALL
set_Tdist_start(POV); // NEED FOR SETTING VARIABLE 'TDIST'
back_r((Node)null, node, 0);
return;
} // end back



/**
 ** method new_back_r     amplify mag going backwards
*/
public double new_back_r (Node caller, Node child, int current_transition_count,
boolean previously_entered_AN) { // returns product of accumulated CS's
int i, size, passed_transition_count=current_transition_count;
double caller_mag, ret_value=0.0, recursion_value=1.0;
Node grand_child;
String caller_name = "(caller_name is null)";
char pol1, pol2; // the polarization symbols for links
boolean entered_AN=false;     // we can start on an AN and then exit a string of
them, that's OK,
                              // so we need to see if we both enter and exit ANs
if (child == null)
       return(0.0);
if (child == Root)
       return(0.0);

String Cn = "  caller="+gui.node_to_string(caller);
String cn = "  child="+gui.node_to_string(child);
String gcn;

if (child.dist <= 2) { // stop near POV
       GUI.P(1,"back_r","OK, stopping near POV at child="+cn);
       return(1);
       }

if (current_transition_count > GUI.max_transition_count) {
// --------------------------------------------
```

```
if (caller.Debug || child.Debug)
        gui.p("nbr: current_transition_count > GUI.max_transition_count:
"+Cn+cn);
        return(0); // Need to return 0 which gets multiplied by prior recursion
values ...
                // want this thread to go backwards with the message 'don't
magnify'
        }

if (!caller.isAN && child.isAN) // Sense going into an AN from a non-AN
        previously_entered_AN=true;

//HERE

size = child.Links.size();
        for (i=0; i<size; i++) {
        grand_child = child.getNodeAtLink(i);
        gcn = "  grand_child="+gui.node_to_string(grand_child);
        if (grand_child==POV)
                {return(1.0);}
        if (grand_child==caller)
                {continue;}
        if (grand_child.dist>-1
                && !grand_child.hasLargerDistThan(child))  // Here WE CHECK ORDER
OF NODES
                {
        entered_AN=previously_entered_AN; // reset
        if (!child.isAN && grand_child.isAN) // Sense going into an AN from a
non-AN
                {
// ---------------------------------------------- Entering AN
                entered_AN=true;
                if (grand_child.Debug) gui.p("nbr: Entering grand_child AN:
"+Cn+cn+gcn);
                }
    if (previously_entered_AN && !grand_child.isAN) // Exiting, don't pursue
into and out of an AN
// ---------------------------------------------- Exiting AN
                {
                if (child.Debug || grand_child.Debug)
                gui.p("nbr: Exiting AN, previously_entered_AN=true, g-child!=AN
"+Cn+cn+gcn);
                continue;
                }
                // RECURSION Here
                pol2 = child.getPol(grand_child);
                if (pol2=='+') {
// ---------------------------------------------- Increment tran-count,
recurse
```

```
                if (grand_child.Debug || child.Debug) gui.p("nbr: + pol and
recursing to grand_child: <"+Cn+cn+gcn);
                recursion_value = new_back_r(child, grand_child,
1+passed_transition_count, entered_AN); // recurse on any, check Type in
beginning
                        }
                else {
// ------------------------------------------------ Just recurse
                if (grand_child.Debug || child.Debug) gui.p("nbr: not + pol but
recursing to grand_child: <"+Cn+cn+gcn);
                recursion_value = new_back_r(child, grand_child,
passed_transition_count, entered_AN); // recurse on any, check Type in beginning
                }
                if (recursion_value > ret_value) { // Notice success reaching
back home
                        if (grand_child.Debug || child.Debug) gui.p("nbr: Success
sensed distal to: <"+Cn+cn+gcn);
                        ret_value = recursion_value;
                        }
                }
}
ret_value *= child.potentiation_mag;
if (ret_value > 0.0) {
// ---------------------------------------------------
        if (caller.Debug || child.Debug) gui.p("nbr: mag'ing child: "+Cn+cn+Cn);
        child.set_mag(ret_value * Node.EMPHASIZED_MAG);
        }
return(ret_value);
} // end new_back_r



/**
 ** method back_r     amplify mag going backwards, // 6/7/99 recurse if type==AN
*/
public double back_r (Node caller, Node child, int current_transition_count) {
// returns product of accumulated CS's
int i, size, passed_transition_count=current_transition_count;
double caller_mag, ret_value=0.0, recursion_value=1.0;
Node grand_child;
String caller_name = "(caller_name  is null)";

if (child == null)
        return(0.0);


if (child.dist <= 1) { // stop near POV
        return(1);
        }
```

```
size = child.Links.size();
for (i=0; i<size; i++) {
    grand_child = child.getNodeAtLink(i);
    if (grand_child==POV)
        return(1.0);
    if (grand_child!=null) {
        if (grand_child.dist>-1
                && grand_child.hasSmallerDistThan(child))  // Here WE CHECK ORDER
OF NODES
                {


// RECURSION Here
                recursion_value = back_r(child, grand_child,
passed_transition_count); // recurse on any, check Type in beginning
                if (recursion_value > 0)
                if (recursion_value > ret_value) {
                        ret_value = recursion_value;
                        }
                }
        }
}

if (caller!=null)
        caller_name = caller.Name;
ret_value *= child.potentiation_mag;
if (ret_value > 0.0) {
child.set_mag(ret_value * Node.EMPHASIZED_MAG);
        }
return(ret_value);
} // end back_r



/**
 ** method threshold_threshold_back_r     amplify mag going backwards, // 6/7/99
recurse if type==AN
*/
public double threshold_back_r (Node caller, Node child) { // returns product of
accumulated CS's
int i, size;
double caller_mag, ret_value=0.0, recursion_value=1.0;
Node grand_child;
String caller_name = "(caller_name  is null)";

if (child == null)
        return(0.0);
```

```
if (child.dist <= 1) { // stop near POV
       return(1);
       }

child.lift_to_threshold();

size = child.Links.size();
for (i=0; i<size; i++) {
    grand_child = child.getNodeAtLink(i);
    if (grand_child==POV)
        return(1.0);
    if (grand_child!=null) {
        if (grand_child.dist>-1
               && grand_child.hasSmallerDistThan(child))  // Here WE CHECK ORDER
OF NODES
               {
// RECURSION Here
               recursion_value = threshold_back_r(child, grand_child); //
recurse on any, check Type in beginning
               }
        }
}
return(ret_value);
} // end threshold_back_r




/**
 **  remember   set Node.min_mag to the current mag (generally, don't allow it
to shrink)
 **
*/
public void remember () {
int i, size;
Node node=null;
Node child=null;

if (gui.lastNode == null)
       return;


GUI.P(0,"remember","lastNode = "+gui.lastNode);
GUI.P(0,"remember","lastNode.Name = "+gui.lastNode.Name);
GUI.P(0,"remember","lastNode.Links.size() = "+gui.lastNode.Links.size());

size = gui.lastNode.Links.size();
for (i=0; i<size; i++) {
       child = (Node)(gui.lastNode.getNodeAtLink(i));
       //if (child.dist == gui.lastNode.dist+1)
```

```
        if (child.dist > gui.lastNode.dist)
              remember_r(gui.lastNode, child);
}

} // end remember


/**
 **  go_until_AN_DN
 **
*/
public void go_until_AN_DN (Node node, boolean did_trans_to_AN) {
int i, size;
Node child;
boolean is_trans_to_AN=false, is_trans_to_DN=false;


GUI.P(0,"go_until_AN_DN", node.Name+", did_trans_to_AN="+did_trans_to_AN);

size = node.Links.size();
for (i=0; i<size; i++) {
        child = node.getNodeAtLink(i);
        if ((child.dist == node.dist+delta_dist) || (child.dist ==
(int)(node.dist)+1))
                {
                if (!node.isAN && child.isAN)
                        is_trans_to_AN = true;
                if (node.isAN && !child.isAN)
                        is_trans_to_DN = true;

                if (did_trans_to_AN && is_trans_to_DN) {
                        GUI.P(0,"go_until_AN_DN", child.Name+", halted:
trans_to_DN=true");
                        break;
                        }
                go_until_AN_DN(child, (is_trans_to_AN || did_trans_to_AN));
                }
        }

} // end go_until_AN_DN



/**
 **  like
 **
*/
public void like (Node node) {
        go_until_DN_AN((Node)null, node, '+');
        go_until_DN_AN((Node)null, node, '-');
} // end like


/**
 **  go_until_DN_AN
 **
*/
public void go_until_DN_AN (Node caller, Node node, char sign) {
int i, size, dist_diff;
Node child;
boolean is_trans_to_AN=false;

if (node == null)
        return;

GUI.P(0,"go_until_DN_AN","direction '"+sign+"',  "+node.Name);
if (caller!=null)
        caller.more_CS(node);



if (sign=='+')
        dist_diff = 1;
else
        dist_diff = -1;



size = node.Links.size();
for (i=0; i<size; i++) {
        child = node.getNodeAtLink(i);
        if ((child.dist == (node.dist+dist_diff)) && child.dist > 2) {
                if (!node.isAN && child.isAN)
                        is_trans_to_AN = true;
                if (is_trans_to_AN) {
                        GUI.P(0,"go_until_DN_AN", child.Name+", halted:
trans_to_AN=true");
                        break;
                        }
                go_until_DN_AN(node, child, sign);
                }
        }
return;
} // end go_until_DN_AN


/**
 **  addCNodeBetweenNodes  :DON'T USE: add a modulating CNode to the link
between node1 and node2
```

```
 **
*/
public void addCNodeBetweenNodes (Node node1, Node node2, Node CNode) { // CSs


Link link = node1.getLinkTo(node2);
link.addCNode(CNode);

} // end addCNodeBetweenNodes


/**
 **  remember_r   set CS between parent and node based on current mag
 **
*/
public void remember_r (Node parent, Node node) {
int i, size;
Node child=null;
double cs, old_cs;

if (parent==null)
       return;
if (node==null)
       return;

cs = parent.get_CS(node);
old_cs = cs;

Link link = parent.getLinkTo(node);
if (link == null) {
       gui.ERROR(0, "remember_r", "link is null to " +node.Name);
       return;
       }

if (node.mag >= Node.MAX_MAG)
       cs *= 1.4;
else if (node.mag >= Node.BIG_MAG)
       cs *= 1.2;
else if (node.mag >= Node.DEFAULT_MAG)
       ;
else
       cs /= 1.2;

link.set_CS(cs);
GUI.P(0,"remember_r",node.Name+".CS changing from "+gui.prec(old_cs,3)+" ->
"+gui.prec(cs,3));

size = node.Links.size();
for (i=0; i<size; i++) {
       child = node.getNodeAtLink(i);
```

```
        //if (child.dist == node.dist+1)
        if ((child.dist == node.dist+delta_dist) || (child.dist ==
(int)(node.dist)+1))
                remember_r(node, child);
        }

} // end remember_r



/**
 **  forget   set Node.min_mag to the current mag (generally, don't allow it to
shrink)
 **
*/
public void forget () {
int i, size;
Node node;
Node child;

size = node_vec.size();
for (i=0; i<size; i++) {
        node = (Node)(node_vec.elementAt(i));
        node.min_mag = node.mag;
        }


} // end forget




/**
 **  showevents
 **
*/
public void showevents () {
int i, size;
Node node;
Node child;

size = node_vec.size();
for (i=0; i<size; i++) {
        node = (Node)(node_vec.elementAt(i));
        if (!node.isEvent) {
                if (null != node.hasSiblingOfType("Event")) {
                if (node.mag < Node.BIG_MAG)
                        node.set_mag(Node.BIG_MAG);
                else
                        node.set_mag(Node.MAX_MAG);
                }
                }
        }
return;
} // end showevents


/**
 **  showtime
 **
*/
public void showtime () {
int i, size;
Node node;
Node child;

mag_r((Node)null, pop.TimeLine, "distal", 3, 0, "+", false);

/****************************************
size = node_vec.size();
for (i=0; i<size; i++) {
        node = (Node)(node_vec.elementAt(i));
        if (node.isEvent) {
                if (node.mag < Node.MAX_MAG)
                        node.set_mag(Node.MAX_MAG);
                }
        }
****************************************/
return;
} // end showtime


/**
 **  peg   set Node.min_mag to the current mag (generally, don't allow it to
shrink)
 **
*/
public void peg () {
int i, size;
Node node;
Node child;

size = node_vec.size();
for (i=0; i<size; i++) {
        node = (Node)(node_vec.elementAt(i));
        node.min_mag = node.mag;
        }


} // end peg
```

```
/*
 * find_AN_named
 */
public Node find_AN_named (String name)
{
int i, size;
Node tn=null, ret_node = null;

size = node_vec.size();
GUI.P(2,"find_AN_named","size="+size+" name="+name);
for (i=0; i<size; i++) {
    tn = (Node)(node_vec.elementAt(i));
    if (tn.Name.equalsIgnoreCase(name) && tn.isAN) {
       GUI.P(2, "DS.find_AN_named", "Found "+name);
       ret_node = tn;
       }
    }
    if (ret_node == null)
       GUI.ERROR(2, "DS.find_AN_named", "Null node for name "+name);
    else
       GUI.P(2, "DS.find_AN_named", "FOUND node for name "+name);

return(ret_node);
} // end find_AN_named


/*
 * find_DN_named
 */
public Node find_DN_named (String name)
{
int i, size;
Node tn=null, ret_node = null;

size = node_vec.size();
GUI.P(2,"find_DN_named","size="+size+" name="+name);
for (i=0; i<size; i++) {
    tn = (Node)(node_vec.elementAt(i));
    if (tn.Name.equalsIgnoreCase(name) && tn.Type.equalsIgnoreCase("DN")) {
       GUI.P(2, "DS.find_DN_named", "Found "+name);
       ret_node = tn;
       }
    }
    if (ret_node == null)
       GUI.ERROR(2, "DS.find_DN_named", "Null node for name "+name);
    else
```

```
        GUI.P(2, "DS.find_DN_named", "FOUND node for name "+name);

return(ret_node);
} // end find_DN_named


/**
 **  moreless    moreless adjust mags to see structure of nodes better, count
num visible
 **
*/
public void moreless (String cmd) {
Node node;

int count=0;
for (int i=0; i<node_vec.size(); i++) { // count them
        if ((node=(Node)(node_vec.elementAt(i))) != null)
                if (node.mag > gui.text_threshold)
                count ++;
        }

if (gui.desired_visible_count == 0) {
        gui.desired_visible_count = count;
        GUI.P(0,"moreless","desired_visible_count initially set to
"+gui.desired_visible_count);
        }

if (cmd.equals("m")) {
                gui.desired_visible_count *= 1.5;
                GUI.P(0,"moreless","desired_visible_count up to
"+gui.desired_visible_count);
                auto_flatten();
        }
else {
                gui.desired_visible_count /= 1.5;
                if (gui.desired_visible_count < 5)
                        gui.desired_visible_count = 5;
                GUI.P(0,"moreless","desired_visible_count down to
"+gui.desired_visible_count);
                auto_flatten();
        }
return;
} // end moreless

/**
 **  auto_flatten    automatically adjust mags to see structure of nodes better,
count num visible
 **
*/
public void auto_flatten () {
```

```
int i, size, count=1000000, iteration_count = 0;
Node tn, node;

while ((count > (gui.desired_visible_count+5)) && (iteration_count++ < 100)) {
// impose a limit of 100 tries
count = 0;
for (i=0; i<node_vec.size(); i++) { // count them
        if ((node=(Node)(node_vec.elementAt(i))) != null)
                if (node.mag > gui.pos_threshold)
                count ++;
        }
GUI.P(0, "DataSea.auto_flatten first-half", iteration_count+") + + "+count+"
nodes > threshold, want "+gui.desired_visible_count);

if (count > gui.desired_visible_count) { // raise threshold
        sharpen();
        }

}

System.err.println("count="+count+", gui.desired_visible_count =
"+gui.desired_visible_count);

// count and decrease the threshold if needed
iteration_count = 0;
while ((count < (gui.desired_visible_count)) && (iteration_count++ < 100)) { //
impose a limit of 100 tries
count = 0;
for (i=0; i<node_vec.size(); i++) { // count them
        if ((node=(Node)(node_vec.elementAt(i))) != null)
                if (node.mag > gui.pos_threshold)
                count ++;
        }
GUI.P(0, "DataSea.auto_flatten second-half", iteration_count+") - - "+count+"
nodes < threshold, want "+gui.desired_visible_count);

// adjust the threshold
if (count < gui.desired_visible_count) { // raise threshold
        flatten();
        }
}
return;
} // end auto_flatten

/**
 **  DataSea.show_node_vec
 **
*/
        public  int show_node_vec () {
        int i;
        for (i=0; i<node_vec.size(); i++)
            if (node_vec.elementAt(i) != null)
            GUI.P(1, "DataSea.show_node_vec",
(((Node)node_vec.elementAt(i)).Name));
        return(0);
}

/**
 **  DataSea.add_to_node_vec
 **
*/
        public static int add_to_node_vec (Node dn) {
        if (node_vec == null)
           node_vec = new Vector(100);
        node_vec.addElement(dn);
// MAYBE NOT A GOOD IDEA TO LINK ROOT TO EVERYTHING  3/24/99
//      if (Root != null)
//             Root.link(dn);

        return(0);
}


/**
 **  DataSea.clear_list    clear the vector used to pass groups of nodes
 **
*/
        public static void clear_list () {
        if (list_vec == null)
           list_vec = new Vector(10);
        else
               list_vec.removeAllElements();

        return;

} // end clear_list


/**
 **  DataSea.add_to_list    add_to the vector used to pass groups of nodes
 **
*/
        public static void add_to_list (Node node) {
        if (list_vec == null)
           list_vec = new Vector(10);
        list_vec.addElement(node);

        return;

} // end add_to_list
```

```
/**
 **  DataSea.stored_into_list_already    clear the vector used to pass groups of
nodes
 **
*/
        public static boolean stored_into_list_already (Node node) {

        if (list_vec.contains(node))
                return(true);
        else
                return(false);

} // end stored_into_list_already




/**
 **  cl      'create_and_link'
 **
*/
        public Node cl (String Name) {
        Node tnode;
        tnode = find_node_named(Name);
        if (tnode == null)
               tnode = cl(Name, "DN", "", Link.DEFAULT_CS);
//      else
//             this.link(tnode);
        return(tnode);
        }

        public Node cl (String Name, double cs) {
        Node tnode;
        // tnode = cl(Name, "DN", "", cs);
        tnode = DataSea.find_node_named(Name);
        if (tnode == null)
               tnode = cl(Name, "DN", "", cs);
        return(tnode);
        }

        public Node cl (String Name, String Type) {
        Node tnode;
        tnode = cl(Name, Type, "", Link.DEFAULT_CS);
        return(tnode);
        }

        public Node cl (String Name, String Type, String Desc) {
```

```
        Node tnode;
        tnode = cl(Name, Type, Desc, Link.DEFAULT_CS);
        return(tnode);
        }

        public Node cl (String Name1, String Type1, String Name2, String Type2) {
        Node node1, node2, tnode;
        node1 = pop.create_node(Name1, Type1);
        node2 = pop.create_node(Name2, Type2);
        node1.link(node2);
//      this.link(node1);
        return(node2);
        }

// new Node sets CS if available into Node, link() sets it into proper CS_R or
CS_L
        public Node cl (String Name, String Type, String Desc, double CS) {
        Node tnode;
        tnode = pop.create_node(Name, Type, Desc);
//      this.link(tnode, Desc, CS);
        return(tnode);
        }


        public Node cl (String Name, String Type, String Desc,
                        int x, int y) {
        Node tnode;
        tnode = cl(Name, Type, Desc, Link.DEFAULT_CS);
        tnode.X = x;
        tnode.Y = y;
        return(tnode);
        }


/****************************************
        public Node cl (String Name, String Type, String Desc,
                        int x, int y, int size_x, int size_y) {
        Node tnode;
        tnode = cl(Name, Type, Desc, Link.DEFAULT_CS);
        tnode.X = x;
        tnode.Y = y;
        tnode.size_x = x;
        tnode.size_y = y;
        return(tnode);
        }

        public Node cl (String Name, String Type, String Desc, String Data,
                        int x, int y, int size_x, int size_y) {
        Node tnode;
        tnode = cl(Name, Type, Desc, Link.DEFAULT_CS);
        tnode.X = x;
        tnode.Y = y;
        tnode.size_x = x;
        tnode.size_y = y;
        tnode.Data[0] = Data;
        return(tnode);
        }
****************************************/

// end cl




/************************************************************************
 **  DataSea.cloner
 **
        public Node cloner (Node dn, String name) {
        Node retdn;
                retdn = new Node();
                retdn.Name = name;
                if (dn != null) {
                        retdn.Name = dn.Name+".";
// maxnodes is also the number of how many nodes have been created.
                retdn.Name = retdn.Name + "(" + maxnodes + ")";
                }
                return(retdn);
                }
        public Node cloner (Node dn) {
                Node retdn;
                retdn = cloner((Node) dn, "NoName");
                return(retdn);
                }
        public Node cloner (String name) {
                Node retdn;
                retdn = cloner((Node) null, name);
                return(retdn);
                }
        public Node cloner () {
                Node retdn;
                retdn = cloner((Node )null, "NoName");
                return(retdn);
                }
*/

/*
 * find_node_named  (overloaded)  First look for AN, then any if not found
 */
```

```
static public Node find_node_named (String name)
{
int i, size;
Node tn=null, ret_node = null;

size = node_vec.size();

// Check for AN first, then anything
if ((ret_node=find_node_named(name, "AN")) == null)
       ret_node=find_node_named(name, (String)null);

return(ret_node);
} // end find_node_named


/*
 * find node named name, with Type of type
 */
static public Node find_node_named (String name, String type)
{
int i, size;
Node tn=null, ret_node = null;

size = node_vec.size();

for (i=0; i<size; i++) {
       tn = (Node)(node_vec.elementAt(i));
       if (type == null) {
              if (tn.Name.equalsIgnoreCase(name))
                     ret_node = tn;
              } else {
              if (tn.Name.equalsIgnoreCase(name)  &&
tn.Type.equalsIgnoreCase(type))
                     ret_node = tn;
              }
       }
return(ret_node);
}  // end find_node_named


/**
 **  create_note    see if it exists, create DN if not, return it
 **                 Automatically create a matching AN for a DN
*/
       public Node create_note (String Desc) {
        int i;
        Node tnode=null;

       GUI.P(0,"create_note", "Desc="+Desc);
```

```
        tnode = new Node("Note", "DN", Desc);
        return(tnode);
} // end create_note

/**
 **  normalize
 **
*/
        static public double normalize () {
         int i, size;
         Node tn;
        Node max_mag_node=null;
        double max_mag=0, normalization_factor=1, delta;


if (!GUI.doNormalization) {
        GUI.P(1,"normalize", "normalization off");
        return(max_mag);
        }

// ------- for all nodes --------
size = node_vec.size();
for (i=0; i<size; i++) {
        tn = (Node)(node_vec.elementAt(i));
        if (tn.mag > 11)
               tn.mag = 10+Math.sqrt(tn.mag-10.0);
        if (tn.mag > max_mag) {
               max_mag = tn.mag;
               max_mag_node = tn;
               }
        }
// ---- end for all nodes -------


if (max_mag < 10.0) {
        GUI.P(1,"normalize", max_mag_node.Name+ ",max_mag="+max_mag+",
normalization_factor="+normalization_factor);
        return(max_mag);
        }

normalization_factor = 10.0 * 1.0/max_mag;

GUI.P(1,"normalize", max_mag_node.Name+ ",max_mag="+max_mag+",
normalization_factor="+normalization_factor);

// ------- for all nodes --------
for (i=0; i<size; i++) {
        tn = (Node)(node_vec.elementAt(i));
        tn.set_mag_no_history(tn.mag * normalization_factor);
        }
```

```
// ---- end for all nodes -------

/*****************************************************************
System.out.println("normalize: log(max_mag)="+Math.log(max_mag));
System.out.println("normalize:
log(normalization_factor)="+Math.log(normalization_factor));
System.out.println("normalize: log(1)="+Math.log(1));
System.out.println("normalize: exp(0)="+Math.exp(0));
System.out.println("normalize: exp(1)="+Math.exp(1));
System.out.println("normalize: log(1.7)="+Math.log(1.7));
System.out.println("normalize: log(3.7)="+Math.log(3.7));
*****************************************************************/

if (POV != null) {
// ------ for children --------------------
size = POV.Links.size();
for (i=0; i<size; i++) {
       tn = (Node)(POV.getNodeAtLink(i));
       if (tn.mag < Node.BIG_MAG)
              tn.set_mag(Node.BIG_MAG);
}
// ----- end for children -----------------
       }

return(max_mag);
} // end normalize


/**
 **  flatten   add constant to everything, and then after normalization, all's
flatter
 **
*/
       public double flatten () {
        int i, size;
        Node tn;
       double max_mag=0.0, delta_mag=0.0, current_mag=0.0, new_mag=0.0;


size = node_vec.size();
// ------- for all nodes --------
for (i=0; i<size; i++) {
       tn = (Node)(node_vec.elementAt(i));
       current_mag = tn.mag;
       delta_mag = (Node.MAX_MAG - current_mag)/10;
       new_mag = current_mag + delta_mag;
       tn.set_mag_no_history(new_mag);
       }
// ---- end for all nodes -------
```

```
GUI.P(0,"flatten", "Done.");
return(max_mag);
} // end flatten


/**
 ** sharpen   subtract constant to everything, and then after normalization,
all's less flat
 **
*/
        public double sharpen () {
         int i, size;
         Node tn;
        double max_mag=0.0, delta_mag=0.0, current_mag=0.0, new_mag=0.0;

size = node_vec.size();
// ------- for all nodes --------
for (i=0; i<size; i++) {
        tn = (Node)(node_vec.elementAt(i));
        current_mag = tn.mag;
        delta_mag = (Node.MAX_MAG - current_mag)/10;
        new_mag = current_mag - delta_mag;
        tn.set_mag_no_history(new_mag);
        }
// ---- end for all nodes -------

GUI.P(0,"sharpen", "Done.");
return(max_mag);
} // end sharpen

/**
 ** find_common_nodes_to      FORGET THIS
 */
 public void find_common_nodes_to (Node node) {
 int i, size;
 Node tn;

size = node.Links.size();
for (i=0; i<size; i++) {
    tn = (Node)(node_vec.elementAt(i));
    if (tn.isAN) {
        GUI.P(0,"find_common_nodes_to"," Got AN named "+tn.Name);
        }
    }
 } // end find_common_nodes_to

/**
 **  DataSea.connect_up
*/
public void connect_up (Node node) {
      Node tn;
      GUI.P(0, "connect_up", "Connecting up node "+node.Name);
   } // end connect_up


   /**
    **  DataSea.add
    **
   */
   public int add_to_unprocessed_vec (Node dn) {
      if (unprocessed_vec == null)
         unprocessed_vec = new Vector(100,10);
      unprocessed_vec.addElement(dn);
      GUI.P(0, "DS.add_to_unprocessed_vec", "added node "+dn.Name+", size="+
      unprocessed_vec.size());
         return(0);
   } // end add_to_unprocessed_vec

   /**
    **  DataSea.PostProcessor
   */
   public void PostProcessor () {
   // add code to have this read a vector of nodes which haven't been processed
   yet,
   // from vector unprocessed_vec, probably added to by 'DataSea.words_input'
      Node tn;

      tn = (Node)(unprocessed_vec.elementAt(0));
      connect_up(tn);
      unprocessed_vec.removeElementAt(0);
   } // end PostProcessor

   /**
    **  DataSea.add_POV
    **  Create a new POV
   */
   public void add_POV () {
   Node tn;
   // add code to have this read a vector of nodes which haven't been processed
   yet,
   // from vector unprocessed_vec, probably added to by 'DataSea.words_input'
          if (POV == null) {
             POV_namer ++;
             POV = new Node("POV"+POV_namer, "POV",
                         "", 0, 0, 20, 10);
            //POV.isPolarized = false;
             POV.z = 0;
            GUI.P(0,"DataSea.add_POV","POV created: "+POV.Name);
             }
           else {
                GUI.P(0,"DataSea.add_POV","POV already exists: "+POV.Name);
                }

} // end add_POV

/**
 **  DataSea.absorb_POV
 **
*/
public void absorb_POV (boolean save) {
Node child = null;

if (POV == null) {
        GUI.P(1,"absorb_POV","POV is null");
        return;
        }

if (save) {
        GUI.P(1,"absorb_POV","Saving the POV as a Context Node");
        if (!POV.Name.substring(0,3).equals("POV")) {
                GUI.WARNING(0,"absorb_POV","POV.Name.substring(0,3) is "
+POV.Name.substring(0,3));
                GUI.ERROR(0,"absorb_POV","POV.Name isn't POVxxx:" +POV.Name);
        } else {
        POV.Name = "P"+POV_namer;
        POV.isPOV = false;
        POV.setType("CN");
        GUI.P(1,"absorb_POV","Setting POV to null, Saved Name is <"+POV.Name+">,
Saved Type is <"+POV.Type+">");
        child = POV.getNodeAtLink(0);
        if (child != null)
                GUI.P(1,"absorb_POV","POV.Link[0] is "+ child.Name);
        else
                GUI.P(1,"absorb_POV","POV.Link[0] is NULL");
        }
}
POV.unlink_all();
if (save) {
        POV.link(child);
        save_links(POV, POV);
        }
   POV = null;
   needdistUpdate=true;
return;
} // end absorb_POV



/**
 **  save_links
 **
*/
public void save_links (Node node, Node CNode) {
int i, size;
Node child;

size = node.Links.size();
for (i=0; i<size; i++) {
        child = node.getNodeAtLink(i);
        if (child.mag > 5) {
                save_links_r(node, child, CNode);
                }
        }
} // end save_links

/**
 **  save_links_r
 **
*/
public void save_links_r (Node node, Node child, Node CNode) {
int i, size;
Node grand_child=null;

if (child.mag <= 5)
        return;

GUI.P(1,"save_links_r","Parent="+node.Name+", Child="+child.Name+",
CNode="+CNode.Name);
addCNodeBetweenNodes(node, child, CNode);
size = child.Links.size();
for (i=0; i<size; i++) {
        grand_child = child.getNodeAtLink(i);
        if ((grand_child.mag > 5) && (grand_child.dist ==  1+child.dist))
                {
                save_links_r(child, grand_child, CNode);
                }
        }
} // end save_links_r


/**
 **  DataSea.more_attraction
 **
*/
        public double more_attraction (boolean inverse_result) {
        if (inverse_result == true)
                gui.magscale /= 1.5;
        else
                gui.magscale *= 1.5;
        return(gui.magscale);
        }

 /**
  **  DataSea.more_repulsion
  **
  **
 */
        public double more_repulsion (boolean inverse_result) {
        if (inverse_result == true)
                gui.ThetaMultiplier -= .1;
        else
                gui.ThetaMultiplier += .1;

        return(gui.ThetaMultiplier);
        }



 /**
  **  set_should_we_pos
  **
 */
        public void set_should_we_pos () {
        int size, tsize, i, j;
         Node tnode, child;

size = node_vec.size();
for (i=0; i<size; i++) {
        tnode = (Node)node_vec.elementAt(i);
        tsize = tnode.Links.size();
        for (j=0; j<tsize; j++) {
                child = tnode.getNodeAtLink(j);
                tnode.setLinks_should_we_pos(child, true); // set link tnode ->
child true
                }
        }
} // end set_should_we_pos



/*
**
**/
void clear_Tdist (){
Node tnode;
int size, i;
     GUI.P(1,"clear_Tdist","Clearing node_vec Tdists.");
    size = node_vec.size();
    for (i=0; i<size; i++) {
        tnode = (Node)node_vec.elementAt(i);
        tnode.Tdist = -1;
        }
return;
} // end clear_Tdist

/*
**
**/
void clear_dist (){
Node tnode;
int size, i;
     GUI.P(1,"clear_dist","Clearing node_vec dists.");
    size = node_vec.size();
    for (i=0; i<size; i++) {
        tnode = (Node)node_vec.elementAt(i);
        tnode.set_dist(-1);
        }
return;
} // end clear_dist

/**
 **  calc_dist_between
 **
*/
public double calc_dist_between (Node node1, Node node2) {
int i, size;
Node tn;

set_Tdist_start(node1);
GUI.P(0,"calc_dist_between"," "+node2.Name+" is "+node2.Tdist+" from
"+node1.Name);
return(node2.Tdist);

} // end calc_dist_between

/**
 **  newtest
 **
*/
public void newtest () {
int i, size;
Node child;


Node ruth = find_node_named("Ruth");

show(ruth);
```

```
Node A1 = find_node_named("A1");
Node A2 = find_node_named("A2");
Node A3 = find_node_named("A3");
Node A4 = find_node_named("A4");
Node B1 = find_node_named("B1");
Node target = find_node_named("target");
if (POV==null) {
        GUI.WARNING(0,"DataSea.newtest","Need a POV");
        return;
        }
set_Tdist_start(target);
System.out.println("A1.dist/Tdist=("+A1.dist+","+A1.Tdist+")");
System.out.println("A2.dist/Tdist=("+A2.dist+","+A2.Tdist+")");
System.out.println("A3.dist/Tdist=("+A3.dist+","+A3.Tdist+")");
System.out.println("B1.dist/Tdist=("+B1.dist+","+B1.Tdist+")");
System.out.println("A4.dist/Tdist=("+A4.dist+","+A4.Tdist+")");
System.out.println("target.dist/Tdist=("+target.dist+","+target.Tdist+")");

} // end newtest

/**
 **  check_for_loop_beginning
 **                              go backwards until we find dist going up
*/
public boolean check_for_loop_beginning (Node caller, Node node) {
int i, size;
Node child=null;
boolean ret=false;

size = node.Links.size();
for (i=0; i<size; i++) {
        child = node.getNodeAtLink(i);
if (child != caller) {
        if (child.dist <= 1) // We've gone all the way back to the POV
                return(ret);
        if (child.dist < node.dist) {
System.out.println("CFLB: (child<node) N("+describe_node_dist(node)+")---
>C("+describe_node_dist(child)+")");
                if (true==check_for_loop_beginning(node, child)) {
                        ret = true;
                }
                node.Tdist = child.Tdist + 1; // if branch has high value of
Tdist, propogate it
                }
        else
        if (child.dist > node.dist) { // May be on branch with reversal inside it
System.out.println("CFLB: (child>node) N("+describe_node_dist(node)+")--
sense_rev-->C("+describe_node_dist(child)+")");
                if (true==sense_rev(node, child)) { // sense_rev will back up and
increase Tdist's
```

```
                        ret = true;
                        node.Tdist = child.Tdist + 1;//continue increasing Tdist's
from sense_rev
System.out.println("CFLB: (child>node) N("+describe_node_dist(node)+")--
increasing Tdist after sense_rev-->C("+describe_node_dist(child)+")");
                }
        }
}
}
return(ret);
} // end check_for_loop_beginning


/**
 **  sense_rev
 **
*/
public boolean sense_rev (Node caller, Node node) {
int i, size;
Node child;
boolean ret = false;

size = node.Links.size();
for (i=0; i<size; i++) {
        child = node.getNodeAtLink(i);
if (child != caller) {
        if (child.dist > node.dist) {
System.out.println("sense_rev: (child>node) N("+describe_node_dist(node)+")---
>C("+describe_node_dist(child)+")");
                ret = sense_rev(node, child);
                if (ret) {  // IF TRUE THEN WE ARE ON THE PATH THAT NEEDS TO BE
CHANGED
                        return(ret);
                        }
                }
        if (child.dist <= node.dist ) { // FOUND REVERSAL POINT, INFORM CALLER
THAT WE DID
System.out.println("sense_rev: (child<=node) N("+describe_node_dist(node)+")
REVERSAL C("+describe_node_dist(child)+")");
                return(true); // don't change this yet
                }
        }
}
return(ret);
} // end sense_rev


// HERE HERE
/**
 **  describe_node_dist
 **
*/
public String describe_node_dist (Node node) {
int i, size;
Node child;
return(node.Name+"("+node.dist+","+node.Tdist+")");
} // end describe_node_dist



/*
**  set_Tdist_start   One-node version
**/
void set_Tdist_start (Node parent){
int level=0;
boolean complete=false, ret;

if (parent==null) {
       GUI.WARNING(0,"set_Tdist_start(1)","parent==null");
       return;
       }
GUI.P(0,"set_Tdist_start(1)","Starting on "+parent.Name);
    clear_Tdist();
    parent.Tdist = 1; // force this first one, then use recursion for others
    level=2;
    while (complete == false) {
        complete = true;
gui.p("Tdist_start: level="+level);
        ret = set_Tdist_recursive(parent, level);
        if (ret == false)
            complete = false;
        level++;
        }
return;
} // end set_Tdist_start



/*
**    set_Tdist_recursive
**                            This increments a node's .Tdist whatever type it
is
**                            Return true only when we've run out of children
**                            or when none of the children are at 1+caller.Tdist
**                            complete starts out true, it conditionally only
changes to false.
**/
boolean set_Tdist_recursive (Node node, int level) { // position this node's
children
int i, size, Tdist;
```

```
boolean ret=true, complete=true;
Node child;
// search for where we were last, where Tdists were set to (level-1)


if (node.Tdist == level) {
        complete=false;
        return(complete);
        }

size = node.Links.size();
for (i=0; i<size; i++) { // check Tdist of all children, position if appropriate
        child = node.getNodeAtLink(i);
        if (child == Root) { // Don't go back through the POV
               return(complete);
               }
        if (child == POV) { // Don't go back through the POV
               return(complete);
               }
        if (child.Tdist == -1) {
                    child.Tdist = ((int)(node.Tdist)+1);
               complete = false;
               }
        else if (child.Tdist > node.Tdist) { // RECURSE
               ret=set_Tdist_recursive(child, level);
               }
        if (ret==false)
               complete=false;
        }

return(complete);
} // end set_Tdist_recursive



/*
**/
void set_dist_start (Node parent){
int level=0;
boolean complete=false, ret;

if (parent == null) {
        GUI.WARNING(0,"set_dist_start(1)"," parent is null");
        return;
        }
GUI.P(0,"set_dist_start(1)","Starting on "+parent.Name);
    clear_dist();
    parent.set_dist(1); // force this first one, then use recursion for others
    parent.set_Tdist(1); // force this first one, then use recursion for others
    level=2;
```

```
        while (complete == false) {
            GUI.P(0,"set_dist_start","iterating on level="+level);
            complete = true;
            // ret = set_dist_recursive(parent, level);
            ret = new_set_dist_recursive(parent, 1, level, 0);
            if (ret == false)
                complete = false;
            level++;
            }
    return;
    } // end set_dist_start

    /*
    **
    **/ // NEW VERSION
    boolean new_set_dist_recursive (Node parent, int current_level, int
    target_level, int AN_level) { // position this parent's children
    int i, size, new_AN_level=AN_level;
    boolean ret=true, complete=true, same_type=false, recurse=false;
    Node child;
    double delta=0.1, new_dist=-1;

    // search for where we were last

    if (current_level >= target_level)
            {
            complete=false;
            return(complete);
            }

    // HERE
    if (POV != null)
    if (parent.StopSpread && parent.dist != (POV.dist+1))  // Don't go back through
    stopped nodes
            return(complete);

    if (parent.isAN)
            parent.AN_level = AN_level;

    // HERE
    size = parent.Links.size();
    for (i=0; i<size; i++)
            {
            child = parent.getNodeAtLink(i);

            ret = true;            // set defaults for next loop
            same_type = false;     // set defaults for next loop
            recurse = false;       // set defaults for next loop

            if (parent.Type == child.Type)
```

```
                        same_type = true;

                if (same_type)
                        new_dist = parent.dist + delta;
                else
                        new_dist = (int)(parent.dist) + 1; // add one to truncated value

                if (child.dist == -1)  { // just beginning
                        child.set_dist(new_dist);
                        child.set_Tdist(parent.Tdist + 1);
                        complete=false;
                        recurse=false; // Done, don't recurse. This slowed me down.
                        }
                else // if already the optimal distance, continue on this path until
        level is met
                if ((child.dist < new_dist + 0.000001) && (child.dist > new_dist -
        0.000001))
                        {
                        recurse=true;
                        }

                if (child.isAN)
                        new_AN_level = AN_level+1;
                if (recurse)
                ret=new_set_dist_recursive(child, current_level+1, target_level,
        new_AN_level);

                if (ret==false) // one instance of ret==false makes the whole thing
        false, we keep on
                        complete=false;

                }

        return(complete);

        } // end NEW VERSION set_dist_recursive



        /*
        **
        **  // MIDDLE VERSION
        boolean set_dist_recursive (Node parent, int level) { // position this parent's
        children
        int i, size, dist;
        boolean ret=true, complete=true, recurse=false;
        Node child;
        double new_dist=-1;

        // search for where we were last, where dists were set to (level-1)
```

```
if (parent.dist == level)
        {
        complete=false;
        return(complete);
        }

// HERE
if (POV != null)
if (parent.StopSpread && parent.dist != (POV.dist+1))  // Don't go back through
stopped nodes
        return(complete);

size = parent.Links.size();
for (i=0; i<size; i++)
{ // check dist of all children, position if appropriate
        child = parent.getNodeAtLink(i);

        if (child.dist == -1) {
                //if (parent.Type == child.Type)
                if (parent.isDN && child.isDN)
                        {new_dist = parent.dist + 0.1;}
                else
                        {new_dist = parent.dist + 1;}
            parent.setLinksVRparmsTo(child);
            complete = false;
            }
        else {
                if (parent.isDN && child.isDN) {
                //if (parent.Type == child.Type)  {}
                if (child.dist >= (parent.dist+0.1)) {
                        {new_dist = parent.dist + 0.1; recurse = true;}
//System.out.println(parent.Name+", "+child.Name+" p.dist="+parent.dist);
                                }
                        }
                else // not same type
                if ((int)(child.dist) >= ((int)parent.dist+1)) {
                        {new_dist = parent.dist + 1; recurse = true;}
                        }
                }

if (child.Name.equalsIgnoreCase("mtg") || child.Name.equalsIgnoreCase("12 Mar
2000")) {
        System.out.println(parent.Name+", "+child.Name+" p.dist="+parent.dist
                +", old_child.dist="+child.dist
                +", new_child.dist="+new_dist+"    recurse="+recurse);
        }
child.set_dist(new_dist);
if (recurse)
```

```
        ret=set_dist_recursive(child, level);
recurse = false; // reset it
if (ret==false)
        complete=false;
}
return(complete);
} // end set_dist_recursive
**/


/*
**
**/ // OLD VERSION
boolean set_dist_recursive (Node parent, int level) { // position this parent's
children
int i, size, dist;
boolean ret=true, complete=true;
Node child;
// search for where we were last, where dists were set to (level-1)

if (parent.dist == level)
        {
        complete=false;
        return(complete);
        }

// HERE
if (POV != null)
if (parent.StopSpread && parent.dist != (POV.dist+1))  // Don't go back through
stopped nodes
        return(complete);

size = parent.Links.size();
for (i=0; i<size; i++)
{ // check dist of all children, position if appropriate
        child = parent.getNodeAtLink(i);

        if (child.dist == -1) {
            child.set_dist((parent.dist+1));
            parent.setLinksVRparmsTo(child);
            complete = false;
            }
        else if (child.dist == (parent.dist+1))
            ret=set_dist_recursive(child, level);
        if (ret==false)
            complete=false;
}
return(complete);
} // end set_dist_recursive
```

```
/**
 **  OKtoFollow     returns boolean whether to invoke action on grand_child,
 **                 looks at three nodes for context
*/
public boolean OKtoFollow (Node parent, Node child, Node grand_child) {
boolean ret_boolean = false;

if (parent == POV) // sometimes POV is too small for test below of mag
       return(true);
if (parent == null)
       return(true);
if (child == null)
       return(true);
if (grand_child == null)
       return(true);

// ((!child.isDN && !grand_child.isDN) || gui.drawDN)  &&
// ((!child.isAN && !grand_child.isAN) || gui.drawAN)  &&
// ((!child.isCN && !grand_child.isCN) || gui.drawCN)  &&
// ((!child.isPN && !grand_child.isPN) || gui.drawPN)  &&
// (!(gui.checkPolarization && parent.isBB && !child.isBB))  &&
// ((!child.isEvent && !grand_child.isEvent) || gui.drawEvent)

if ( child.mag >= 0.2 )
       ret_boolean = true;

return(ret_boolean);

} // end OKtoFollow



/*
** set_ChildNum_start   Set the ChildNum variable, by calling
set_ChildNum_recursive
**/
void set_ChildNum_start (Node parent, Node node, boolean theta_org_flag){
int level=0;

if (node == null)
       return;

GUI.P(1,"set_ChildNum_start","Starting on "+node.Name);
set_ChildNum_recursive(parent, node, theta_org_flag);
return;
} // end set_ChildNum_start

/*
** set_ChildNum_recursive   We assume the dist has been properly set, now we
**                      just count the number of children having dist =
node.dist+1
**                      Returns max number of children to the caller, node
**/
void set_ChildNum_recursive (Node parent, Node node, boolean theta_org_flag) {
// set the children's ChildNum
int i, j, size, ChildNum=0, tChildNum=0, ChildCount;
int x, y;
Node child, tn;
Node node1, node2;

int BigChildCount = 0;

String parentName = "unassigned";

if (parent != null)
        parentName = parent.Name;


if (gui.Debug==2) {
if (parent==null)
System.out.println("sCNr: set_ChildNum() START: "+node.Name);
else
System.out.println("sCNr: "+parent.Name+"->"+node.Name);
}

//
// Calculate BigChildCount based on all siblings (rather than distal children)
size = node.Links.size();
for (i=0; i<size; i++)
        {
        child = node.getNodeAtLink(i);
        if (child.mag > Node.BIG_MAG)
                BigChildCount++;
        }
node.BigChildCount = BigChildCount;


for (i=0; i<size; i++)
        {
        child = node.getNodeAtLink(i);
        if (((child.dist > node.dist)) && child!=Root) // This child'll be drawn
next
                {
        //      if (ChildNum < Node.MAX_CHILDREN)
                {
                //      if (OKtoFollow(parent, node, child))
                                {
                                node.child_vec[ChildNum] = i; // set starting with
0
```

```
                                        child.ChildNum = ++ChildNum;  // starts with 1
                                        set_ChildNum_recursive(node, child,
theta_org_flag);
                                        }
                                }
                        }
                }
node.ChildCount = ChildNum;
ChildCount = node.ChildCount;

if (ChildCount >= Node.MAX_CHILDREN) {
        GUI.WARNING(0,"set_ChildNum_recursive","ChildCount >= Node.MAX_CHILDREN,
bailing.");
        return;
        }

/*****************************/
if (theta_org_flag) {
// Now, order the vector child_vec
for (j=0; j< ChildCount-2; j++)
for (i=0; i<ChildCount-1-j; i++) {

// SWAP AND LATER RESET CHILDNUM
node1 = node.getNodeAtLink( node.child_vec[i] );
node2 = node.getNodeAtLink( node.child_vec[i+1] );

if (node1.mag < node2.mag) {
        x = node.child_vec[i];
        node.child_vec[i] = node.child_vec[i+1];
        node.child_vec[i+1] = x;
        x = node1.ChildNum;
        node1.ChildNum = node2.ChildNum;
        node2.ChildNum = x;
//      GUI.P(1,"set_ChildNum_recursive","j("+j+")swapping nodes
"+node.child_vec[i].Name+" <-> "+node.child_vec[i+1].Name);
        }
}
} // end theta_org_flag

for (i=0; i<ChildCount; i++) {
        node1 = node.getNodeAtLink( node.child_vec[i] );
        node1.ChildNum = i+1; // ChildNum starts with 1, the index with 0
}
/*****************************/

return;
} // end set_ChildNum_recursive


/**
```

```
 ** method calc_mags
 ** find avg of two largest neighbors' mags
*/
public  void calc_mags (Node target_node) {
        double mag1=0, mag2=0, avg_mag=0;
        Node tn, node1, node2;
        int size, i;

size = target_node.Links.size();
switch (size) {
        case 0: break; // ought not to happen
        case 1: avg_mag = 0.5*(target_node.getNodeAtLink(0)).mag;
                if (avg_mag > target_node.mag)
                        target_node.set_mag(avg_mag);
                break;
        case 2: mag1 =  (target_node.getNodeAtLink(0)).mag;
                mag2 =  (target_node.getNodeAtLink(1)).mag;
                target_node.set_mag(0.5*(mag1 + mag2));
                break;
        default:
                for (i=0; i<size; i++) { // find avg of two largest mag's
                        tn = target_node.getNodeAtLink(i);
                        if (tn.mag > mag1)
                                mag1 = tn.mag;
                        else if (tn.mag > mag2)
                                mag2 = tn.mag;
                        }
                avg_mag = 0.5*(mag1 + mag2);
                if (avg_mag > target_node.mag)
                        target_node.set_mag(avg_mag);
                break;
        }
GUI.P(1,"calc_mags","node '"+target_node.Name+"' mag="+target_node.mag);
} // calc_mags



/**
 **  backup
 **
*/
public void backup () {
int i, size;
Node parent, tparent;

if (gui.lastNode == null)
        return;

parent = gui.lastNode;
```

```
do {
        tparent = parent.getParent();
        if (tparent==null)
                break;
        if (tparent.dist > 1)          // don't go back to the POV,
                parent = tparent;      // but do allow going back one beyond the
same type
        if (tparent.Type != gui.lastNode.Type)
                break;
        } while (true);

// now, the parent is prior to the last one of the same type as the starting
point
gui.lastNode = parent;

} // end backup



/*
 **  potmag   potentiate distal to target, mag distal to POV
 **           spread initially both polarizations, halt at pol-transition
 **           touch !ANs but don't recurse.
 **           Then, mag distally from another high-order AN.
 */
public  void potmag (String words[], int num_words) {
int size, i;
Node pot_target=null, mag_target=null;


if (num_words == 1)
        pot_target = GUI.lastNode;
else
if (num_words == 2) {
        pot_target = find_node_named(words[1]);
        }
else
if (num_words == 3) {
        pot_target = find_node_named(words[1]);
        mag_target = find_node_named(words[2]);
        }

if (pot_target == null) {
        GUI.WARNING(0,"potmag", "Need a lastNode");
        return;
        }

if (pot_target == null) {
        GUI.WARNING(0,"potmag", "Need a name or at least a lastNode. POV sibling
is implied.");
        return;
        }

if (POV != null) {
        if (mag_target == null)  // if no second arg is given, assume its the
first relative of POV
                mag_target = POV.getNodeAtLink(0);
        }
if (mag_target == null) {
        GUI.WARNING(0,"potmag", "Need a mag_target");
        return;
        }
System.err.println("----- Beginning of potmag() --------------------");
GUI.P(0,"potmag","Using pot_target "+pot_target.Name+", mag_target of
"+mag_target.Name);
potmag(pot_target, mag_target);
System.err.println("----------- End of potmag() --------------------");
return;
} // end potmag


public  void potmag (Node pot_target, Node mag_target){
// SET THE POTENTIATION DISTAL FROM TARGET NODE (using Tdist)
        /***************************
        **      set_Tdist_start(pot_target); // NEED FOR SETTING VARIABLE 'TDIST'
        **      pot(pot_target, true); // USES 'TDIST'
        ***************************/
        poth(pot_target); // sets Tdist from Thesaurus_node
// SET THE MAG DISTAL FROM PRIMARY SIBLING OF POV
        magall(mag_target); // USES 'DIST'

return;
} // end potmag


public void poth (Node node) {  //
int size, i;
double depth, up_depth, down_depth;
Node tn=null;

if (node == null)
        return;

if (Thesaurus_node == null) {
        GUI.WARNING(0,"poth","Need Thesaurus_node. Aborted.");
        return;
        }
```

```
set_Tdist_start(Thesaurus_node); // NEED FOR SETTING VARIABLE 'TDIST'

// Check current depth of node
depth = node.Tdist;
up_depth = depth / 2;
down_depth = 100; // go all the way down always

gui.p("pot("+gui.node_to_string(node)+")");
node.set_pot(GUI.current_TS);

// SET THE POTENTIATION DISTAL FROM SECONDARY NODE
size = node.Links.size();
for (i=0; i<size; i++) {
        tn = node.getNodeAtLink(i);
// SEE IF CHILD IS DOWNHILL
        if ((tn.Tdist == 1+node.Tdist)) { // going downhill ...
                poth_r(tn, "down", down_depth);
                }
        else
// OR UPHILL
        if ((tn.Tdist == -1+node.Tdist)) { // going uphill ...
                poth_r(tn, "up", up_depth);
                }
        }

return;
} // end poth

/**
 **  poth_r
 **
*/
public void poth_r (Node node, String direction, double target_depth) {
int i, size, delta;
Node child;

if (node.Debug)
        GUI.P(0,"poth_r","setting pot on "+node.Name +", Tdist="+node.Tdist+",
Type="+node.Type);

node.set_pot(GUI.current_TS);

// Bail out if we are at the target depth
if (node.Tdist == target_depth) {
        return;
        }

if (direction.equalsIgnoreCase("up"))
        delta = -1;
else
```

```
if (direction.equalsIgnoreCase("down"))
        delta = 1;
else {
        GUI.WARNING(0, "poth_r","Wrong argument for direction passed, is
"+direction);
        return;
        }

size = node.Links.size();
for (i=0; i<size; i++) {
        child = node.getNodeAtLink(i);
        if (node.Tdist + delta == child.Tdist) // recurse if in the correct
direction
                poth_r(child, direction, target_depth);
        }

} // end poth_r


/**
 **  lower_all_pots
 **
public void lower_all_pots (Node node) {
int i, size;
Node child;

size = node_vec.size();
for (i=0; i<size; i++) {
        child = (Node)(node_vec.elementAt(i));
        child.set_pot(GUI.current_TS);
        }

} // end lower_all_pots
*/

/**
 ** method pot    2nd version distributes pot_r considering polarization of
start
*/
public  void pot (String words[], int num_words) {
Node node=null;

if (num_words == 1)
        node = gui.lastNode;
else
if (num_words > 1)
        {
        node = find_node_named(words[1]);
        }
```

```
if (node == null) {
        GUI.WARNING(0,"pot", "Need a name or at least a lastNode");
        return;
        }

GUI.P(1,"pot", "Top-level, Running on "+node.Name);

//lower_all_pots();
pot(node, false);  // if given as command, don't recurse
return;
} // end pot


public  void pot (Node node, boolean recurse) {  // distributes pot_r
considering polarization of start
int size, i;
Node tn=null;

if (node == null)
        return;


// SET THE POTENTIATION DISTAL FROM SECONDARY NODE
size = node.Links.size();
for (i=0; i<size; i++) {
        tn = node.getNodeAtLink(i);
        tn.set_pot(GUI.current_TS);
        GUI.P(0,"pot","on "+tn.Name
                +", Tdist="+tn.Tdist+", Type="+tn.Type);
        if (recurse && (tn.Tdist > node.Tdist)) { // recurse only if child.Tdist
> us
                pot_r(tn, node.getPol(tn), tn.Type);  // pass on the polarization
                }
        }
return;
} // end pot


public  void pot_r (Node node, char original_pol, String original_type) {
int size, i;
Node tn=null;

if (node == null)
        return;

// SET THE POTENTIATION DISTAL FROM SECONDARY NODE
size = node.Links.size();
for (i=0; i<size; i++) {
        tn = node.getNodeAtLink(i);
        tn.set_pot(GUI.current_TS);
        if ((tn.Tdist == 1+node.Tdist)
                && (tn.Type.equals(original_type))
                //&& (original_pol==node.getPol(tn))
                ) { // recurse
                GUI.P(1,"pot_r",node.getPol(tn)+" Type="+tn.Type
                        +" Tdist="+tn.Tdist+" "+node.Name+" calling "+tn.Name);
                pot_r(tn, original_pol, original_type);
                }
                else { // not recursing
                //GUI.P(1,"pot_r","NOT RECURSING "+node.getPol(tn)+"
Type="+tn.Type
                //      +" Tdist="+tn.Tdist+" "+node.Name+" calling "+tn.Name);
                }
        }
return;
} // end pot_r


/**
 ** method back_p
*/
public  void back_p (String words[], int num_words) {
Node node=null;

        if (num_words==0)
                return;

        if (num_words==1)
                node = gui.lastNode;
        else
                node = find_node_named(words[1]);

        if (node == null)
                return;

        GUI.P(0,"back_p", "Operating on "+node.Name);
        back_p(node); // recursive fn

} // end back_p (String, int)

public  void back_p (Node node) {
int size, i;
Node tn=null;


size = node.Links.size();
for (i=0; i<size; i++) {
        tn = node.getNodeAtLink(i);
        if (tn.dist < node.dist) {
                GUI.P(0,"back_p", " i is "+i+", tn="+tn.Name);
                tn.set_pot(GUI.current_TS);
                // tn.potentiation_TS = GUI.current_TS; // p'_mag determined from
this
                back_p(tn); // recurse
                }
        }
} // end back_p

/**
 ** method upurls  magnify the URLS upstream from each URL
*/
public  void upurls (String words[], int num_words) {
        int target_level=0, size, i, child_size, child_i;
        Node node=null, child;
        Link link=null;
        boolean only = false;// do the target_level and distal ones

gui.P(0,"upurls","Begun.");

size = node_vec.size();
for (i=0; i<size; i++) {
        node = (Node)(node_vec.elementAt(i));
        if (node.isURL) {
                // ------ for children --------------------
                child_size = node.Links.size();
                for (child_i=0; child_i<child_size; child_i++) {
                        child = (Node)(node.getNodeAtLink(child_i));
                        if (child.isURL) {
                                link = node.getLinkTo(child);
                                if (child == link.NodeL) {
                                        gui.P(0,"upurls",child.Name+" magnified by
"+node.Name);
                                        //child.more_mag(); // FINALLY, MAG' THE
HIGHER URL
                                        child.set_mag(child.mag * node.mag); //
FINALLY, MAG' THE HIGHER URL
                                        }
                        }
                }
                // ----- end for children ----------------

                }
        }
gui.P(0,"upurls","Done.");
return;
} // end upurls


/**
```

```
 **  set_selected_on_big_nodes
 **
*/
public void set_selected_on_big_nodes () {
int i, size;
Node child;

size = node_vec.size();
for (i=0; i<size; i++) {
        child = (Node)(node_vec.elementAt(i));
        if (child.mag >= Node.MED_MAG + 0.1)
                child.isSelected = true;
        }
return;
} // end set_selected_on_big_nodes



/**
 **  selectrecent   set isSelected of all AN's touched within 10 seconds
 **
*/
public void selectrecent () {
int i, size;
Node child;

size = node_vec.size();
for (i=0; i<size; i++) {
        child = (Node)(node_vec.elementAt(i));
        if (child.isAN && (child.TS > GUI.current_TS-10000))
                child.isSelected = true;
        }
return;
} // end selectrecent


/**
 **  flattenANs   set all ANs to MED_MAG
 **
*/
public void flattenANs () {
int i, size;
Node child;

gui.P(0,"flattenANs","Begun.");

size = node_vec.size();
for (i=0; i<size; i++) {
        child = (Node)(node_vec.elementAt(i));
        if (child.isAN)
```

```
                child.set_mag(Node.MED_MAG);
        }

gui.P(0,"flattenANs","Done.");

return;
} // end flattenANs



/**
 **  ans      show the ANs two links from lastNode, force their mag to
MAX_MAG+0.1
 **
*/
public void ans () {
int i,j, size_i, size_j;
Node child, grand_child;


Node node = gui.lastNode;

size_i = node.Links.size();
for (i=0; i<size_i; i++) {
        child = node.getNodeAtLink(i);
                size_j = child.Links.size();
                for (j=0; j<size_j; j++) {
                        grand_child = child.getNodeAtLink(j);
                        if (grand_child.isAN)
                                grand_child.set_mag(Node.MAX_MAG+0.1);
                        }
        }

} // end ans


/**
 **  inhstored
 **
*/
public void inhstored () {
int i, size;
Node node;

gui.P(0,"inhstored", "Begun.");

if (big_mag_node_vec == null) {
        gui.WARNING(0,"inhstored","big_mag_node_vec is null.");
        return;
        }
```

```
size = big_mag_node_vec.size();
for (i=0; i<size; i++) {
        node = (Node)(big_mag_node_vec.elementAt(i));
        node.set_mag(Node.SMALL_MAG);
        }
gui.P(0,"inhstored", "Lowered "+big_mag_node_vec.size()+" elements in
big_mag_node_vec.");
return;
} // end inhstored


/**
 **  storebig
 **
*/
public void storebig () {
int i, size;
Node child;

big_mag_node_vec = null; // how do we release an object? 'free()' doesn't seem
to work
big_mag_node_vec = new Vector(10);

size = node_vec.size();
for (i=0; i<size; i++) {
        child = (Node)(node_vec.elementAt(i));
        if (child.isAN && child.mag > Node.BIG_MAG)
                big_mag_node_vec.addElement(child);
        }
gui.P(0,"storebig", "There are "+big_mag_node_vec.size()+" elements in
big_mag_node_vec.");
return;
} // end storebig


/**
 ** method absURLs
*/
public  void absURLs () {  // Assumes user has run selectrecent() and
flattenANs().
                        // Mag unselected AN's of URLs based on the URL's mag
                        // Assume that large-mag AN's have been selected (based
                        // perhaps on their being touched recently)
int size, i;
Node node;

GUI.P(0,"absURLs", "Begun.");

// ------- for all nodes --------
```

```
size = node_vec.size();
for (i=0; i<size; i++) {
       node = (Node)(node_vec.elementAt(i));
       if (node.isURL)
              if (node.mag > Node.BIG_MAG) {
                     //set_Tdist_start(node); // NEED FOR SETTING VARIABLE
'TDIST'
                     GUI.P(0,"absURLs", "Starting on URL <"+node.Name+">
mag="+node.mag);
                     absURLs_r((Node)null, node);
                     //prime_or_mag(node);
                     }
       }
// ---- end for all nodes -------

//needdistUpdate = true;
return;
} // end absURLs


/**
 **  prime_or_mag   Either prime this node (make it sensitive to another call)
or mag it
 **                    Called usually by fn meant to characterize DNs
*/
public void prime_or_mag (Node node) {
int i, size;
Node child;

GUI.P(0,"prime_or_mag","Called on "+node.Name+", dist="+node.dist);

// Call from xabs() from URL, find AN children, invoke this fn on them

size = node.Links.size();
for (i=0; i<size; i++) {
child = node.getNodeAtLink(i);

if (node.isURL)
if (child.isAN)
if (child.potentiation_TS>(GUI.current_TS-1000)) { // we've been touched
recently
       child.more_mag(node);
       System.out.println("node.isURL & child.isAN & touched <"+node.Name+"> to
<"+child.Name+">, dist="+child.dist);
       prime_or_mag(child);
}
else {
       child.set_pot();
       System.out.println("node.isURL & child.isAN & NOT touched <"+node.Name+">
to <"+child.Name+">, dist="+child.dist);
```

```
        }

if (node.isAN)
if (child.isAN)
if (node.goesUpstreamTo(child))
if (child.potentiation_TS>(GUI.current_TS-1000)) { // we've been touched
recently
        child.more_mag(node);
        System.out.println("node.isAN & child.isAN & upstream & touched
<"+node.Name+"> to <"+child.Name+">, dist="+child.dist);
        prime_or_mag(child);
        }
else {
        child.set_pot();
        System.out.println("node.isAN & child.isAN & upstream & NOT touched
<"+node.Name+"> to <"+child.Name+">, dist="+child.dist);
        }

}


/*******************************************************
//
//if ((node.isAN && node.goesUpstreamTo(child))||(!node.isAN &&
child.dist>node.dist)){
//      if (child.potentiation_TS>(GUI.current_TS-1000)) { // we've been touched
recently
//      if (child.isAN) {
//              GUI.P(1,"prime_or_mag","Mag'ing & recursing from "+node.Name+" to
"+child.Name+", dist="+child.dist);
//              child.more_mag();
//              prime_or_mag(child);
//              }
//      else
//      GUI.P(1,"prime_or_mag","NOT AN going from "+node.Name+" to
"+child.Name+", dist="+child.dist);
//      }
//      else  {
//              GUI.P(1,"prime_or_mag","BAD TS, pot'ing going from "+node.Name+"
to "+child.Name+", dist="+child.dist);
//              child.set_pot();
//              }
//      }
//else
//              GUI.P(1,"prime_or_mag","REJECTING going from "+node.Name+" to
"+child.Name+", dist="+child.dist);
*******************************************************/

} // end prime_or_mag
```

```
/**
 ** method absURLs_r    increment node.mag if node.Type=="AN", recurse only if
child is AN
*/
public  void absURLs_r (Node caller, Node node) {
        int size, i;
        Node tn;
        String caller_name = "<blank>";

// DON'T DO ANYTHING IF a caller is AN and child is not AN.
if (caller != null) {
        caller_name = caller.Name;
        if (caller.isAN && !node.isAN) {
                GUI.P(1,"absURLs_r","BAIL: (caller.isAN && !node.isAN)
caller="+caller.Name+", node="+node.Name);
                return;
                }
}

// DECIDE IF WE SHOULD MAG THIS
if (node.isAN) {
        GUI.P(0,"absURLs_r","MAGGING: "+(node.Name
                +"              ").substring(0,10)
                +"caller="+caller_name+"(dist="+caller.dist+") prime_or_mag()" );
        caller.more_mag(node);
}

// DECIDE IF WE SHOULD RECURSE ON PARENT OF node
size = node.Links.size();
for (i=0; i<size; i++) {
        tn = node.getNodeAtLink(i);
        GUI.P(1,"absURLs_r","TESTING <"
                +node.Name+"->"
                +tn.Name);
        if (
        (node.isAN && tn.isAN) // if parent=AN, then to recurse, child must be AN
        || !node.isAN          // if parent !AN, then child can be anything
        ) {
        if (tn.dist<node.dist) {
                GUI.P(1,"absURLs_r","RECURSING: from <"
                        +node.Name+">("+node.dist+") to "
                        +tn.Name+"("+tn.dist+")");
                absURLs_r(node, tn);
                }
        else
                GUI.P(1,"absURLs_r","NOT-RECURSING: from <"
                        +node.Name+">("+node.dist+") to "
                        +tn.Name+"("+tn.dist+")");
        }
else
        GUI.P(1,"absURLs_r","NOT-RECURSING (AN): from <"
                +node.Name+">("+node.dist+"),isAN="+node.isAN+", to "
                +tn.Name+"("+tn.dist+"),isAN="+tn.isAN);
}
return;
} // end absURLs_r


/**
 **  connect_all_to_POV
 **
*/
public void connect_all_to_POV () {
int i, size;
Node child;

if (POV==null)
        return;

set_Tdist_start(POV); // NEED FOR SETTING VARIABLE 'TDIST'

size = node_vec.size();
for (i=0; i<size; i++) {
        child = (Node)(node_vec.elementAt(i));
        if (child.mag > gui.relations_threshold) {
                gui.p(gui.node_to_string(child));
                threshold_back_r((Node)null, child);
                }
        }



} // end connect_all_to_POV


/**
 ** method abs
*/
public  void abs (String words[], int num_words) {
        int target_level=0;
        Node node;
        boolean only = false;// do the target_level and distal ones

        if (num_words==0)
                return;

        if (num_words==1)
                node = gui.lastNode;
        else
```

```
                    node = find_node_named(words[1]);

               if (node == null) {
                    gui.WARNING(0,"abs","node is null");
                    return;
                    }

               if (words[0].equalsIgnoreCase("abs"))
                    target_level = 5;
               if (words[0].equalsIgnoreCase("abs1")) {
                    target_level = 1;
                    only = true;
                    }
               if (words[0].equalsIgnoreCase("abs2")) {
                    target_level = 2;
                    only = true;
                    }
               if (words[0].equalsIgnoreCase("abs3")) {
                    target_level = 3;
                    only = true;
                    }
               if (words[0].equalsIgnoreCase("abs4")) {
                    target_level = 4;
                    only = true;
                    }
               if (words[0].equalsIgnoreCase("abs5")) {
                    target_level = 5;
                    only = true;
                    }
               if (words[0].equalsIgnoreCase("abs6")) {
                    target_level = 6;
                    only = true;
                    }



               abs(node, target_level, only);

     connect_all_to_POV();


               } // end abs



     /**
      ** method abs
     */
     public  void abs (Node node, int target_level, boolean only) {
     int size, i;
```

```
Node tn;
boolean recurse=false;

if (node == null)
        return;
if (node == Root)
        return;
node.lift(4); // keep start high


GUI.P(0,"abs", "Operating on "+node.Name);

// set_Tdist_start(node); // NEED FOR SETTING VARIABLE 'TDIST'
size = node.Links.size();
for (i=0; i<size; i++) {
        tn = node.getNodeAtLink(i);
        recurse = false; // reset
        if (((tn.dist==(int)node.dist+1)||(tn.dist==node.dist+delta_dist)) ){
                recurse = true;
                if (tn.isAN && node.isAN) { // if both AN, go upstream only
                        if (node.goesUpstreamTo(tn))
                                recurse &= true;
                        else
                                recurse = false;
                        }
                }
        else
                recurse = false;
        if (recurse) {
                abs_r(node, tn, target_level, 1, only); // start with
this_level=0
                }
        GUI.P(1,"abs", "NOT recursing from "+node.Name+" to "+tn.Name);
}
return;
} // end abs

/**
 ** method abs_r
 **        increment node.mag if node.Type=="AN", recurse only if child is AN
*/
public  void abs_r (Node caller, Node node, int target_level, int this_level,
boolean only) {
int size, i;
Node tn;
boolean recurse=false;

        if (this_level > target_level)
                return;
        if (node.isAN) {
              if (only) { // only increase if exactly on target_level
                    if (this_level == target_level)
                          node.more_mag(caller);
                    }
              else {
                    node.more_mag(caller);
              }
              this_level++;
              }

        if (this_level > target_level)
              return;

        size = node.Links.size();
        for (i=0; i<size; i++) {
              recurse = false; // reset
              tn = node.getNodeAtLink(i);
              if (
              (node.isAN && tn.isAN) // if parent=AN, then to recurse, child
must be AN
              || !node.isAN         // if parent !AN, then child can be
anything
              ) {

              if (((tn.dist==(int)node.dist+1)
              ||(tn.dist<=node.dist+delta_dist+0.00001
              &&tn.dist>=node.dist+delta_dist-0.00001)
                    ) ) {
                    recurse = true;
                    if (tn.isAN && node.isAN) { // if both AN, go upstream
only
                          if (node.goesUpstreamTo(tn)) {
                                recurse &= true;
                                }
                          else {
                                recurse = false;
                                }
                          }
                    }
              else {
                    recurse = false;
                    }
              if (recurse) {
                    abs_r(node, tn, target_level, this_level, only);
                    }
        }
}
return;
```

```
} // end abs_r


/**
 **  count_distal   uses a 'global' variable, returns it: also used globally by
functions
 **
*/
public int count_distal (Node node) {
int return_value;

        GlobalDistalCount = 0;
        count_distal_r(node);

        return_value = GlobalDistalCount;
GUI.P(1,"count_distal","Found "+return_value+" distal nodes.");
        return(return_value);

} // end count_distal

/**
 **  count_distal_r
 **
*/
public void count_distal_r (Node node) {
int i, size;
Node child;

if (node == null)
        return;

size = node.Links.size();
for (i=0; i<size; i++) {
        GlobalDistalCount += size;
        child = (Node)(node.getNodeAtLink(i));
        if (child.dist > node.dist)
                count_distal_r(child);
        }

} // end count_distal_r

/**
 ** method sides   chain data nodes by making those of same type in a chain
visible
*/
public void sides (Node node) {
int i, size;
Node tn;

if (node.isAN) {
```

```
size = node.Links.size();
for (i=0; i<size; i++) {
        tn = (Node)(node.getNodeAtLink(i));
        sides_r(tn, tn.Type);
        }
}
else
        sides_r(node, node.Type);

} // end sides

/**
 ** method sides_r   chain data nodes by making them all visible
*/

public void sides_r (Node node, String type) {
Node child=null;
int i, size;

if (node == null)
        return;
if (node == Root)
        return;

// RECURSE
size = node.Links.size();
for (i=0; i<size; i++) {
        child = node.getNodeAtLink(i);
// check for dis-similar type to magnify
        if ((child.dist > node.dist) && (!child.Type.equalsIgnoreCase(type))) {
                pump(child);
                gui.getToolkit().sync(); // drama
                //gui.show_node_once(child);
        }
// check for similar type to recurse on
        if ((child.dist > node.dist) && (child.Type.equalsIgnoreCase(type))) {
System.err.println("sides_r,'"+node.Name+"'-> recursing on "+child.Name+" from
"+node.Name);
                sides_r(child, type);
        }
}
return;
} // end sides_r



/**
 **  shiftIn
 **
```

```
*/
public void shiftIn (String[] words, int num_words) {
Node CNode=null;

if (null == (CNode = figure_out_node("shiftIn", words, num_words)))
        return;
gui.P(0,"shiftIn","Running on CNode<"+CNode.Name+">");
mag_CSs_of_CNode(CNode);//mag CS's and mag
} // end shiftIn


/**
 **  shiftOut
 **
*/
public void shiftOut (String[] words, int num_words) {
Node CNode=null;

if (null == (CNode = figure_out_node("shiftOut", words, num_words)))
        return;
gui.P(0,"shiftOut","Running on CNode<"+CNode.Name+">");
inh_CSs_of_CNode(CNode);//mag CS's and mag
} // end shiftOut


/**
 **  shiftOutAll
 **
*/
        public void shiftOutAll (String[] words, int num_words) {
         int size, j_size, i, j;
         Node node;
        Link siblink;

size = node_vec.size();
for (i=0; i<size; i++) {
    node = (Node)(node_vec.elementAt(i));
        j_size = node.Links.size();
        for (j=0; j<j_size; j++) {
                siblink = node.getLink(j);
                if (i<10)
                        GUI.P(0,"DataSea.shiftOutAll","resetting CS to
"+Link.MIN_CS+
                                " for node <"+node.Name+"> and
<"+siblink.NodeR.Name+">");
                if (siblink!=null)      {
                        siblink.set_CS(Link.MIN_CS);
                        }
                }
        }
```

```
} // end shiftOutAll


/**
 **  shiftInAll
 **
*/
        public void shiftInAll (String[] words, int num_words) {
         int size, j_size, i, j;
         Node node;
        Link siblink;

size = node_vec.size();
for (i=0; i<size; i++) {
    node = (Node)(node_vec.elementAt(i));
        j_size = node.Links.size();
        for (j=0; j<j_size; j++) {
                siblink = node.getLink(j);
                if (i<10)
                        GUI.P(0,"DataSea.shiftInAll","resetting CS to
"+Link.MED_CS+
                                " for node <"+node.Name+"> and
<"+siblink.NodeR.Name+">");
                if (siblink!=null)     {
                        siblink.set_CS(Link.MED_CS);
                        }
                }
        }
} // end shiftInAll


/*
 * mag_CSs_of_CNode    magnify nodes connected to the Context Node of 'node'
 */
static public Node mag_CSs_of_CNode (Node CNode)
{
int i, size;
Link link=null;
Node child=null, ret_node = null;

CNode.more_mag();

size = CNode.ContextLinks.size();
//gui.P(0,"mag_CSs_of_CNode","Running on "+CNode.Name+" with "+size+" links.");
for (i=0; i<size; i++) {
        link = (Link)CNode.ContextLinks.elementAt(i);
// Set CS to at least the DEFAULT for those related to this CNode
       if ((link.CS_R < Link.DEFAULT_CS) ||  (link.CS_L < Link.DEFAULT_CS))
              link.set_CS(Link.DEFAULT_CS);
       else
              link.more_CS();

// Set mag of nodes linked by this link to at least MED_MAG
       if (link.NodeR.mag < Node.MED_MAG)
              link.NodeR.set_mag(Node.MED_MAG);
       else
              link.NodeR.more_mag();
       if (link.NodeL.mag < Node.MED_MAG)
              link.NodeL.set_mag(Node.MED_MAG);
       else
              link.NodeL.more_mag();

       //gui.P(0,"mag_CSs_of_CNode","Just increased mags and CS_R&L of link
between <"+link.NodeR.Name+"> and <"+link.NodeL.Name+">");
}

//gui.P(0,"mag_CSs_of_CNode","Done.");
return(ret_node);
} // end mag_CSs_of_CNode


/*
 * inh_CSs_of_CNode    inhibit nodes connected to the Context Node of 'node'
 */
static public Node inh_CSs_of_CNode (Node CNode)
{
int i, size;
Link link=null;
Node child=null, ret_node = null;

size = CNode.ContextLinks.size();
gui.P(0,"inh_CSs_of_CNode","Running on <"+CNode.Name+"> with "+size+" links.");
for (i=0; i<size; i++) {
       link = (Link)CNode.ContextLinks.elementAt(i);
       link.less_CS();
       }
return(ret_node);
} // end inh_CSs_of_CNode


/*
 * set_CSs_of_CNode    set CSs of links connected to the Context Node of 'node'
 */
static public Node set_CSs_of_CNode (Node CNode, double CS)
{
int i, size;
```

```
Link link=null;
Node child=null, ret_node = null;

size = CNode.ContextLinks.size();
for (i=0; i<size; i++) {
       link = (Link)CNode.ContextLinks.elementAt(i);
       link.CS_R = CS;
       link.CS_L = CS;
       }
return(ret_node);
} // end set_CSs_of_CNode


/**
 ** method local  local data nodes by making those of same type in a local
visible
*/
static public void local (Node node) {
int i, size;
Node child;

if (node == null) {
       gui.WARNING(0,"local","No lastNode available");
       return;
       }


node.set_mag(Node.MAX_MAG);
gui.P(1,"local","Running on "+node.Name);

size = node.Links.size();
for (i=0; i<size; i++) {
       child = (Node)(node.getNodeAtLink(i));
       child.set_mag(child.mag + Node.DELTA_MAG);
       gui.P(1,"local","Running on "+child.Name);
       }

} // end local

/**
 ** method chain_  chain data nodes by making those of same type in a chain
visible
*/
public void chain (Node node) {
int i, size;
Node child;

if (node == null) {
       gui.WARNING(0,"chain","No lastNode available");
       return;
```

```
        }

size = node.Links.size();
for (i=0; i<size; i++) {
        child = (Node)(node.getNodeAtLink(i));
        chain_r(child, child.Type, "distal");
        chain_r(child, child.Type, "proximal");
        }
} // end chain


public void chain_r (Node node, String type, String direction) {
Node child=null;
int i, size;

if (node == null)
        return;
if (node == Root)
        return;

gui.P(1,"chain_r","Running on "+node.Name);

// RECURSE
size = node.Links.size();
for (i=0; i<size; i++) {
        child = node.getNodeAtLink(i);
// check for similar type
        if ((child.dist > node.dist) && direction.equals("distal") &&
child.Type.equalsIgnoreCase(type)) {
                chain_r(child, type, direction);
                gui.getToolkit().sync(); // drama
                gui.getToolkit().sync(); // drama
                gui.getToolkit().sync(); // drama
                child.more_mag(node);
        }
        else
        if ((child.dist < node.dist) && direction.equals("proximal") &&
child.Type.equalsIgnoreCase(type)) {
                chain_r(child, type, direction);
                gui.getToolkit().sync(); // drama
                gui.getToolkit().sync(); // drama
                gui.getToolkit().sync(); // drama
                child.more_mag(node);
        }
}
return;
} // end chain_r
```

```
/**
 ** method pump  pump up all distal nodes
*/
public void pump (Node node) {
int i, size;
Node child;

if (node == null)
       node = Root;

node.set_mag(Node.MAX_MAG-.2);

size = node.Links.size();
for (i=0; i<size; i++) {
       child = (Node)(node.getNodeAtLink(i));
       if ((child.dist > node.dist) && (child.Type.equalsIgnoreCase(node.Type)))
              pump_r(node, child);
       }

} // end pump

/**
 ** method pump_r   pump data nodes by making them all visible
*/
public void pump_r (Node parent, Node node) {
Node child=null;
int i, size;

if (node == null)
       return;
if (node == Root)
       return;
//if (node.isAN || parent.isAN)
//     return;

node.set_mag(Node.MAX_MAG-.2, parent); // allow use of CS from parent to node

gui.P(1,"pump_r","Running on "+node.Name+" from caller "+parent.Name);

// RECURSE
size = node.Links.size();
for (i=0; i<size; i++) {
       child = node.getNodeAtLink(i);
       if ((child.dist > node.dist) && (child.Type.equalsIgnoreCase(node.Type)))
              pump_r(node, child);
       }
return;
} // end pump_r
```

```
/**
 ** method clean   strip data nodes by stripping away DNs making them barely
visible
*/
public void clean () {
Node node;

node = gui.lastNode; // try the lastNode first
if (null == null) // try the Root next
       node = Root;

System.out.println("clean(), node is "+node.Name);
clean(node);

} // end clean

/**
 ** method clean   strip data nodes by stripping away DNs making them barely
visible
*/
public void clean (Node node) {
       if (node == null) {
              gui.WARNING(0,"clean","Given node is null.");
              return;
              } else
              gui.P(0,"clean","Running on "+node.Name);

       reset_mags(Node.BARELY_VISIBLE_MAG);
       mag_ans((Node)null, node, 2, 0);
       mag_ans((Node)null, node, 1, 0);
       node.set_mag(Node.MAX_MAG);
       //clean_r(node, count_distal(node));
} // end this version of clean


/**
 ** method clean   strip data nodes by stripping away DNs making them barely
visible
*/
public void clean (String[] words, int num_words) {
Node node=null;

if (num_words==1)
       node = gui.lastNode;
else
       node = find_node_named(words[1]);

clean(node);

} // end clean
```

```
/**
 ** method clean_r   strip away data nodes making them barely visible
*/
public void clean_r (Node node, int distal_count) {
Node child=null;
int i, size;
int delay=0;

if (node == null)
       return;

// Scale how long we delay based on how many nodes we'll operate on
// We want to take about 2 seconds for everything
if (distal_count > 0)
       delay = 10000/distal_count;

// RECURSE
       size = node.Links.size();
       for (i=0; i<size; i++) {
              child = node.getNodeAtLink(i);
              if (child.dist > node.dist) {
                     child.set_mag(Node.BARELY_VISIBLE_MAG);
                     clean_r(child, distal_count);
              }
       }
return;
} // end clean_r


/**
 **  mag_ans  This mags AN's which are a certain number of AN links away
3/6/2000
 **
*/
public void mag_ans (Node caller, Node node, int target_level, int
current_level) {
int i, size;
Node child=null;

if (node == null)
       return;

//System.out.println("  ");
//System.out.print("   ["+current_level+"]   ");
if (caller != null)
       if (!caller.isAN && node.isAN)
              if (caller.goesUpstreamTo(node)) {
                     //System.out.print("  [++upstream++]   ");
```

```
                        if (current_level == target_level-1) {
if (node.TS_diff() > 0) {
                                System.out.print("  caller<"+caller.Name+">-
>set_mag<"+node.Name+">, current_level="+current_level);
                                if (node.mag < Node.MAX_MAG)
                                        node.set_mag(Node.MAX_MAG);
                                else
                                        node.more_mag();
                                }
                                }
                        current_level ++;
                        }
                else {
                //System.out.print("  caller<"+caller.Name+"> not_upstream -
><"+node.Name+">");
                return;
                }

if (current_level >= target_level)
        return;

size = node.Links.size();
for (i=0; i<size; i++) {
        child = node.getNodeAtLink(i);
        if (child == Root) ;
        else
        if (child == caller) ;
        else
        if (child.dist > node.dist) { // is distal, no 'else'
        if (node.isAN) {
                if (!child.isAN)
                        ;
                else    { // child is AN also
                        if (node.goesUpstreamTo(child)) // no real 'else'
                                {
                                //System.out.print("  [upstream]  ");
                                if (current_level == target_level-1) {
if (child.TS_diff() > 0) {
//System.out.print("  node<"+node.Name+">->SET_MAG<"+child.Name+">,
current_level="+current_level);
                                        System.out.print("
SET_MAG<"+child.Name+">, "+"current_level="+current_level);
                                        if (node.mag < Node.MAX_MAG)
                                                child.set_mag(Node.MAX_MAG);
                                        else
                                                child.more_mag();
                                        }
                                        }
                                if (current_level < target_level) // recurse
```

```
                                        mag_ans(node, child, target_level,
1+current_level);
                                    }
                              }
                        }
                        else    // node is not an AN, see if child is
                              mag_ans(node, child, target_level, current_level);
                        }
                  }
      return;
      } // end mag_ans


      /**
       **  heavyans  This mags AN's which are a certain number of AN links away
      3/6/2000
       **
      */
      public void heavyans (Node caller, Node node, int target_level, int
      current_level) {
      int i, size;
      Node child=null;

      if (node == null)
            return;

      //System.out.println("  ");
      //System.out.println("  ["+current_level+"]  ");
      if (caller != null)
            if (!caller.isAN && node.isAN)
                  if (caller.goesUpstreamTo(node)) {
                        //System.out.print("  [++upstream++]  ");
                        if (current_level == target_level-1) {
                              System.out.print("  -
MAG(+"+node.Links.size()+")<"+node.Name+">-  ");
                              node.set_mag(node.mag + node.Links.size());
                              }
                        current_level ++;
                        }

      if (current_level >= target_level)
            return;

      //if (caller != null)
      //System.out.print("<"+caller.Name+">-><"+node.Name+">");
      //else
      //System.out.print("<NULL>-><"+node.Name+">");

      size = node.Links.size();
      for (i=0; i<size; i++) {
        child = node.getNodeAtLink(i);
        if (child == Root) ;
        else
        if (child == caller) ;
        else
        if (child.dist > node.dist) { // is distal, no 'else'
        //System.out.print("  <"+child.Name+"> dist+++  ");
        if (node.isAN) {
                if (!child.isAN)
                        ;
                else    { // child is AN also
                        if (node.goesUpstreamTo(child)) // no 'else'
                                {
                                //System.out.print("  [upstream]  ");
                                if (current_level == target_level-1) {
                                System.out.print("
MAG(+"+node.Links.size()+")<"+node.Name+">  ");
                                        child.set_mag(node.mag +
node.Links.size());
                                        }
                                if (current_level < target_level) // recurse
                                        heavyans(node, child, target_level,
1+current_level);
                                }
                        //else
                                //System.out.print("  [downstream]  ");
                        }
                }
                else    // node is not an AN, see if child is
                        heavyans(node, child, target_level, current_level);
                }
        //else
        //System.out.print("  <"+child.Name+"> dist---  ");
        }
return;
} // end heavyans



/**
 **  enhance    enhance !ANs based on number of their !AN-children
 **
*/
public void enhance (String[] words, int num_words) {
int i,j, size, node_size, child_counter, node_counter=0;
Node node, child;
String strj="", Type="UNINITIALIZED TYPE";
double new_mag;

if (num_words == 1) {
        Type = "URL";
        gui.P(0,"enhance","Default Type is "+Type);
        }
else {
        Type = words[1];
        gui.P(0,"enhance","Type is "+Type);
        }


size = node_vec.size();
for (i=0; i<size; i++) {
        node = (Node)(node_vec.elementAt(i));
        if (node.Type.equalsIgnoreCase(Type)) {
                child_counter = 0;
                node_size = node.Links.size();
                for (j=0; j<node_size; j++) { // NOW, COUNT CHILDREN OF CORRECT
TYPE
                        child = (Node)(node.getNodeAtLink(j));
                        if (child.Type.equalsIgnoreCase(Type))
                                child_counter++;
                        }
                if (child_counter>1) { // NOW, INCREASE THE MAG OF THE NODE
'child'
                        new_mag = node.mag+0.3*child_counter;
System.out.print(node.Name+":"+node.mag+"->"+new_mag+"           ");
                        node.set_mag(new_mag);
                        node_counter++;
                        }
                }
        }

gui.P(0,"enhance", node_counter+" nodes enhanced.");

return;
} // end enhance

/**
 **  choices    list the ANs above BIG_MAG for each level of dist from Thes
 **
*/
public void choices () {
int i,j, size, counter;
Node child;
String strj="";

gui.clear_global_str();
gui.add_to_global_str("Choices: ");

for (j=2;j<8;j++) {
        strj="";
```

```
                counter = 0;
                gui.add_to_global_str("Depth "+j+":");
                size = node_vec.size();
                for (i=0; i<size; i++) {
                        child = (Node)(node_vec.elementAt(i));
                        if (child.isAN && child.mag>Node.BIG_MAG && child.dist==j) {
                                counter ++;
                                strj=strj+"          <"+child.Name+">";
                                }
                        }
                        if (counter > 1)
                                gui.add_to_global_str(strj);
                        else
                        if (counter == 1)
                                gui.add_to_global_str("        (only 1-node)");
                        else
                                gui.add_to_global_str("          (empty)");
        }

        } // end choices



        /**
         **  magtype   mag directly linked nodes of same type with given direction.
         **
        */
        public void magtype (Node node, String type, char direction) {

        int i, size;
        Node child;

        if (node==null)
                return;

        if (type==null)
                type = node.Type;

        if (direction==' ')
                direction = 'd';

        // ------ for children --------------------
        size = node.Links.size();
        for (i=0; i<size; i++) {
                child = (Node)(node.getNodeAtLink(i));
                if (child.Type.equalsIgnoreCase(type))
                        if (((direction=='d') && (child.dist > node.dist))
                        ||
                        ((direction=='p') && (child.dist < node.dist)))
                        {
```

```
                        child.more_mag(node);
                        magtype(child, type, direction);
                        }
}
// ----- end for children -----------------
return;
}

/**
 **  magall
 **
*/
public void magall (Node node) {

mag_r(null, node, "distal", gui.INFINITE_DEPTH, 0, "+", true);
mag_r(null, node, "distal", gui.INFINITE_DEPTH, 0, "+", true);

} // end magall


/**
 **  magd
 **
*/
public void magd () {

if (gui.lastNode != null)
mag_r(null, gui.lastNode, "downstream", gui.INFINITE_DEPTH, 0, "+", true);

} // end magd


/*
/**
 **  inhd
 **
*/
public void inhd () {

if (gui.lastNode != null)
mag_r(null, gui.lastNode, "downstream", gui.INFINITE_DEPTH, 0, "-", true);

} // end inhd


/*
 * mag    Full version
 */
public void mag (String[] words, int num_words, String direction, int max_dist,
String more_or_less) {
```

```
Node node=null;

if (num_words==1)
        node = gui.lastNode;
else
        node = find_node_named(words[1]);


if (node != null) {
        mag_r((Node)null, node, direction, max_dist, 0, more_or_less, false);
        }
else
        {
        GUI.WARNING(0,"DataSea.mag","Can't get a node, num_words="+num_words);
        }
} //end mag


/*
 * mag  simplest version
 */
public void mag (Node node) {
        mag_r((Node)null, node, "distal", 5, 0, "+", false);
} // end mag

/*
 * mag  simplest version    more_or_less is either "+" or "-"
 */
public void mag (Node node, String direction, String more_or_less) {
        mag_r((Node)null, node, direction, 5, 0, more_or_less, false);
} // end mag (simplest)

/*
 * mag  simplest version    more_or_less is either "+" or "-"
 */
public void mag (Node node, String direction, String more_or_less, int dist) {
        mag_r((Node)null, node, direction, dist, 0, more_or_less, false);
} // end mag (simplest)


/**
 ** method mag_r    7-variable version
*/
public void mag_r (Node node, Node child, String direction, int max_call_depth,
int this_call_depth, String more_or_less, boolean cross_AN_DN_boundary) {
Node grand_child=null;
int i, size;
boolean distal=false, proximal=false, both=false, OK=false;
boolean downstream=false, upstream=false;
```

```
if (child == null)
      return;
if ((node != null) && (child == Root)) // OK to mag root if there's no caller
      return;


if (child.isCN) {
      if (gui.drawCN) {
            if (more_or_less.equalsIgnoreCase("+"))
                  mag_CSs_of_CNode(child);
            else
                  inh_CSs_of_CNode(child);

            }
      }
// HANDLE MODE
// SET UP THE MODES
if (direction.equalsIgnoreCase("downstream"))
      downstream = true;
if (direction.equalsIgnoreCase("upstream"))
      upstream = true;
if (direction.equalsIgnoreCase("distal"))
      distal = true;
if (direction.equalsIgnoreCase("proximal"))
      proximal = true;
if (direction.equalsIgnoreCase("both"))
      both = true;


// INCREMENT this_call_depth      refers to number of recursive calls, not
Node.dist
if (++this_call_depth > max_call_depth)
      return;


//System.err.println("mag_r: "+this_call_depth+" < "+max_call_depth);


// HANDLE THIS_DIST
// INCREASE the mag of the child arg'
if (more_or_less.equalsIgnoreCase("+")) {
if (child.TS_diff() > 0) { // this TS is before GUI.lastCommandTS
//System.err.println("mag_r: depths:"+this_call_depth+" < "+max_call_depth+">,
more_mag("+child.Name+")");
      child.more_mag(node);
      }
      }
else {
      child.less_mag(node);
```

```
        }


// RECURSE
        size = child.Links.size();
        for (i=0; i<size; i++) {
                grand_child = child.getNodeAtLink(i);
                if (proximal && (grand_child.dist < child.dist))
                        OK = true;
                else
                if (both)
                        OK = true;
                else
                if (distal && (grand_child.dist > child.dist))
                        OK = true;
                else
                if (downstream && child.getPol(child)=='+')
                        OK = true;
                else
                if (upstream && child.getPol(child)=='-')
                        OK = true;

                // sense AN-DN transition:
                // Don't recurse by passing through AN to DN: DN's refer to other
contexts
                // dist=2 means that child is connected to starting point
                if (node != null)
                if ((this_call_depth >= 2) && (child.isAN && !grand_child.isAN)
&& !cross_AN_DN_boundary) {
                        OK = false;
                        }
                if (grand_child.StopSpread)
                        OK = false;
                if (OK) {
                        if (distal || both || downstream)
                        mag_r(child, grand_child, "distal", max_call_depth,
this_call_depth, more_or_less, cross_AN_DN_boundary);
                        if (proximal || both || upstream)
                        mag_r(child, grand_child, "proximal", max_call_depth,
this_call_depth, more_or_less, cross_AN_DN_boundary);
                        OK = false;
                        }
                }
return;
} // end mag_r


// CLEAN UP FUNCTIONS

/**
```

```
 **  simplify   hide DNs distal to ANs
 **
*/
public void simplify (String[] words, int num_words) {
int i, size;
Node node, tn;

        if (num_words==1)
                node = gui.lastNode;
        else
                node = find_node_named(words[1]);

// CHECK FOR ERRORS
        if (node==null) {
        if (num_words>1)
             GUI.WARNING(0,"DataSea.simplify","Can't find node "+words[1]);
        else
             GUI.WARNING(0,"DataSea.simplify","Neither Name given nor existing
lastNode.");
                return;
                }

GUI.P(0,"DataSea.simplify","Working from "+node.Name);
simplify_recursive((Node)null, node);

} // end simplify


/**
 **  simplify_recursive  inhibit DNs following ANs
 **
*/
public void simplify_recursive (Node caller, Node child) {
int i, size;
Node tn;

if (caller == null) {
size = child.Links.size();
for (i=0; i<size; i++) {
        tn = (Node)(child.getNodeAtLink(i));
        if (tn.dist > child.dist)
                simplify_recursive(child, tn);
        }
}
else
if (caller.isAN && child.isDN)
        inhibit(child);
else
{
size = child.Links.size();
for (i=0; i<size; i++) {
        tn = (Node)(child.getNodeAtLink(i));
        if (tn.dist > child.dist)
                simplify_recursive(child, tn);
        }
}

} // end simplify_recursive


/**
 **  strip_r  marginalize all non-AN's after first AN distal to node
 **
*/
public void strip_r (Node parent, Node node) {
int i, size;
Node child;
boolean inhibit_it = false;

if (node == null)
        return;

size = node.Links.size();
for (i=0; i<size; i++) {
        child = (Node)(node.getNodeAtLink(i));
        if (child.dist > node.dist) {
                if (parent != null)
                        if (parent.isAN && !child.isAN) // check AN-!AN boundary
                                inhibit_it = true;
                if (inhibit_it)
                        inhibit(child);
                else
                        strip_r(node, child);
                }
        }
} // end strip_r


/**
 **  strip  marginalize all non-AN's after first AN distal to node
 **
*/
public void strip (String[] words, int num_words) {
Node node=null;

if (num_words==1)
        node = gui.lastNode;
else
        node = find_node_named(words[1]);
```

```
strip_r((Node)null, node);
} // end strip


/**
 ** inhibit   put below threshold the distal nodes, and the proximal nodes up
to first AN
 **          this version handles words
*/
public void inhibit (String[] words, int num_words) {
int i,j, size, size_j;
Node node, child, tn;


// HANDLE IDENTIFYING THE CORRECT NODE TO START ON
       if (num_words==1)
              node = gui.lastNode;
       else
              node = find_node_named(words[1]);

// CHECK FOR ERRORS
       if (node==null) {
       if (num_words>1)
           GUI.WARNING(0,"DataSea.inhibit","Can't find node "+words[1]);
       else
           GUI.WARNING(0,"DataSea.inhibit","Neither Name given nor existing
lastNode.");
              return;
              }
       else
              GUI.P(0,"inhibit","Found node named '"+node.Name+"'");

// -------------------------------------------------------------
gui.show_node_once(node);
inhibit(node);
} // end inhibit

/*
 ** inhibit  single arg version, inhibit proximally and distally from node
 **
*/
public void inhibit (Node node) {
inhibit_distally(node);
// inhibit_proximally(node);
} // end inhibit

/*
 ** inhibit_distally  put below threshold the distal nodes, and the proximal
nodes up to first AN
 **
```

```
*/
public void inhibit_distally (Node node) {
int i, size;
Node child=null;

node.less_mag();
GUI.shrinking_allowed = false;

size = node.Links.size();
for (i=0; i<size; i++) {
        child = (Node)(node.getNodeAtLink(i));
        if (child.dist > node.dist)
                inhibit_distally(child); // decrease all distal nodes
        }
GUI.shrinking_allowed = true;

return;
} // end inhibit_distally


/*
 **  inhibit_proximally  reduce proximal nodes up to first AN
 **
*/
public void inhibit_proximally (Node node) {
int i, size;
Node parent=null;

if (null == POV)  // go backwards and if AN is found, decrease distal from it
        return;

GUI.P(0,"inhibit_proximally","node="+node.Name);

size = node.Links.size();
for (i=0; i<size; i++) {
        parent = (Node)(node.getNodeAtLink(i));
        if (parent.dist < node.dist) {
        if (parent.isAN && (parent.dist <= 2)) {
         GUI.P(0,"inhibit_proximally","breaking for AN parent="+parent.Name+"
dist="+parent.dist);
                break;
                }
        if (parent.dist > 2) {
                parent.less_mag();
                GUI.shrinking_allowed = false;
// RECURSE
                inhibit_proximally(parent);
                GUI.shrinking_allowed = true;
                }
                } // if parent's dist < node's
        }
} // end inhibit_proximally


/**
 **  vote_branch
 **      spread mag changes (+ or -, depending on String s)
 **      both proximally and distally for positive, distal only for negative
 **
*/
        public void vote_branch (String[] words, int num_words, String s) {
        Node node = null;

        if (num_words==1)
                node = gui.lastNode;
        else
                node = find_node_named(words[1]);

// CHECK FOR ERRORS
        if (node==null) {
        if (num_words>1)
             GUI.WARNING(0,"vote_branch","Can't find node "+words[1]);
        return;
        }
        else
        vote_branch(node, s);
} //  end vote_branch


        public void vote_branch (Node node, String s) {
         int i;
        String saved_spread_mode = gui.mode_obj.spread_mode;

if (node == null)
        node = gui.lastNode;
if (node==null)
        return;

GUI.P(0,"vote_branch","Acting '"+s+"', on Node="+node.Name);

if (s.equals("+")) { // PLUS
        gui.mode_obj.spread_mode = "both";
        mag(node, "both", "+");
        }
if (s.equals("-")) { // MINUS
        gui.mode_obj.spread_mode = "distal";
        GUI.shrinking_allowed = false;
        strip_r((Node)null, node);
        node.set_mag(gui.text_threshold - 0.1);
        GUI.shrinking_allowed = true;
        }
gui.mode_obj.spread_mode = saved_spread_mode;
return;
} // end vote_branch



/**
 ** method sim     amplify similar nodes to target (same type, linked ANs)
 **                if started on AN, call on non-AN children
*/
        public void sim (String[] words, int num_words, char sign) {
        int i, size;
         Node node, child;

// HANDLE IDENTIFYING THE CORRECT NODE TO START ON
        if (num_words==1)
                node = gui.lastNode;
        else
                node = find_node_named(words[1]);

// CHECK FOR ERRORS
        if (node==null) {
        if (num_words>1)
             GUI.WARNING(0,"DataSea.sim"+sign,"Can't find node "+words[1]);
        else
             GUI.WARNING(0,"DataSea.sim"+sign,"Neither Name given nor existing
lastNode.");
                return;
                }
        else
                GUI.P(0,"sim"+sign,"Found node named '"+node.Name+"'");

        if (node.isURL) {
                spread_url_sim(node, sign);
                }

        if (node.isAN) { // call on all children, no recursion really
                size = node.Links.size();
                for (i=0; i<size; i++) {
                        child = node.getNodeAtLink(i);
                        if (!child.isAN)
                                sim(child, sign);
                        }
                return;
                }
        else
                sim(node, sign);
```

```
node.set_mag(Node.MAX_MAG); // Make sure we don't normalize this one down

        return;
} // end sim

/**
 ** method sim     amplify similar nodes to target (same type, linked ANs)
 **                    Looks only at child of directly connected ANs, of same
type as 'node'
*/
        public void sim (Node node, char sign) {
        int i, j, i_size, j_size;
        Node child, grand_child;

// HANDLE IDENTIFYING THE CORRECT NODE TO START ON
// -----------------------------------------------------------

spread_url_sim(node, sign);

i_size = node.Links.size();
for (i=0; i<i_size; i++) {
child = node.getNodeAtLink(i);
if (child.isAN)
        {
        GUI.P(0,"sim"+sign, "Type=AN, I_size="+i_size+", "+node.Name+"["+i+"] is
"+child.Name);
        j_size = child.Links.size();
for (j=0; j<j_size; j++) {
        grand_child = child.getNodeAtLink(j);

// DON'T MAG THE AN's ... only the non-ANs
/*******************************************************************************
        if (sign == '+')
                child.a_little_more_mag(); // mag the intervening AN
        else
                child.a_little_less_mag(); // inhibit the intervening AN
*******************************************************************************/

// If the type of grand_child is the same as the called node, let's mag it ...
        if ((grand_child.Type.equals(node.Type)||(grand_child.isAN)) &&
grand_child!=node)
        {
        GUI.P(0,"sim"+sign, node.Name+"("+node.Type+") ->
"+child.Name+"("+child.Type+") -> "+ grand_child.Name+"("+grand_child.Type+")");
        if (sign == '+')
                grand_child.a_little_more_mag(); // mag the grand_child ...
        else
                grand_child.a_little_less_mag(); //  ... or inhibit the
grand_child
        if (grand_child.isURL)
```

```
                        spread_url_sim(grand_child, sign);
                }
                }
                }
        }
        // -----------------------------------------------------------

        return;
        } // end sim



        /**
         **  spread_url_sim
         **
        */
        public void spread_url_sim (Node node, char sign) {
        int i, size, j, j_size;
        Node child, grand_child=null;

        size = node.Links.size();
        for (i=0; i<size; i++) {
                child = node.getNodeAtLink(i);
                if (child.isURL) {
                        GUI.P(0,"spread_url_sim"+sign,
                                node.Name+"("+node.Type+") ->
"+child.Name+"("+child.Type+") Pol="
                                +node.getPol(child));
                        if (sign == '+')
                                child.more_mag();
                        else
                                child.less_mag();
                        if ('-' == node.getPol(child)) {
                                j_size = child.Links.size();
                                for (j=0; j<j_size; j++) {
                                grand_child = child.getNodeAtLink(j);
                                if (grand_child.isURL) {
                                if ('+' == child.getPol(grand_child)) {
                                GUI.P(0,"spread_url_sim"+sign,
                                        "2nd-level:"+child.Name+"("+child.Type+")-
>"+grand_child.Name+"("
                                        +grand_child.Type+") Pol="
                                        +child.getPol(grand_child));
                                        if (sign == '+')
                                                grand_child.more_mag();
                                        else
                                                grand_child.less_mag();
                                                }
                                                }
                                }
                }
            }
        }

        } // end spread_url_sim

        /**
         **  tickle
         **
        public void tickle (Node node) {
        int i, size;
        Node child;

        size = node.Links.size();
        for (i=0; i<size; i++) {
                child = node.getNodeAtLink(i);
                if (child.isURL) {
                        GUI.P(0,"tickle", node.Name+" tickling " +child.Name);
                        child.more_mag();
                        }
                }
        } // end tickle
        */


        /**
         ** method unsim     decrease similar nodes to target (same type, linked ANs)
                public void unsim (String[] words, int num_words) {
                int i, j, i_size, j_size;
                 Node node, child, grand_child;

        // HANDLE IDENTIFYING THE CORRECT NODE TO START ON
                if (num_words==1)
                        node = gui.lastNode;
                else
                        node = find_node_named(words[1]);

        // CHECK FOR ERRORS
                if (node==null) {
                if (num_words>1)
                    GUI.WARNING(0,"DataSea.unsim","Can't find node "+words[1]);
                else
                    GUI.WARNING(0,"DataSea.unsim","Neither Name given nor existing
        lastNode.");
                        return;
                        }
                else
                        GUI.P(0,"unsim","Found node named '"+node.Name+"'");

        GUI.shrinking_allowed = false;
```

```
// ------------------------------------------------------------
gui.show_node_once(node);
i_size = node.Links.size();
        for (i=0; i<i_size; i++) {
        child = node.getNodeAtLink(i);
        if (child.isAN && child!=node) {
        j_size = child.Links.size();
        for (j=0; j<j_size; j++) {
                grand_child = child.getNodeAtLink(j);
                if (grand_child.Type.equals(node.Type) && grand_child!=node)
                        grand_child.less_mag(child); // de-emphasize ANs
                }
        }
}
// ------------------------------------------------------------
GUI.shrinking_allowed = true;

return;
} // end unsim
*/

/**
 **  isolate
 **
*/
        public void isolate () {
         int size, links_size, i,j;
         Node node;
        Link link=null;
        double max_CS=0;

size = node_vec.size();
GUI.P(0,"isolate","Scanning <"+size+"> nodes.");
for (i=0; i<size; i++) {
        node = (Node)(node_vec.elementAt(i));
        // DETERMINE THE MAXIMUM CS FROM ALL THE LINKS TO THIS NODE
        max_CS = get_max_CS(node);
// NOW SET THE MAG PROPORTIONAL TO THE MAXIMUM CS TO THIS NODE
        System.out.print(node.Name+"<"+gui.prec(max_CS,3)+"> ");
        if (max_CS < 0.9*Link.DEFAULT_CS)
                node.set_mag(max_CS/Link.DEFAULT_CS);
        }
} // end isolate


/**
 **  get_max_CS
 **
*/
public double get_max_CS (Node node) {
```

```
int i, size;
Link link;
double max_CS=0;

size = node.Links.size();
for (i=0; i<size; i++) {
        link = (Link)node.Links.elementAt(i);
        if (max_CS < link.CS_R)
                max_CS = link.CS_R;
        if (max_CS < link.CS_L)
                max_CS = link.CS_L;
        }

return(max_CS);
} // end get_max_CS



/**
 **  halve_mags
 **
*/
        public void halve_mags () {
         int size, i;
         Node tn;

size = node_vec.size();
GUI.P(0,"halve_mags","Halving all mags ... size="+size);
for (i=0; i<size; i++) {
    tn = (Node)(node_vec.elementAt(i));
        tn.set_mag(tn.mag * 0.5);
        }
} // end halve_mags


/**
 **  drill_kernel   print list of distal ANs
 **
*/
public void drill_kernel (Node caller) {
int i, size;
Node child;

size = caller.Links.size();
for (i=0; i<size; i++) {
        child = (Node)(caller.getNodeAtLink(i));
        if (child.dist > caller.dist) {
                if (caller.isDN && child.isAN) {  // dim'distal DNs after ANs
                        gui.global_str[0] = gui.global_str[0] +",
"+child.Name+"="+caller.Name;
```

```
                        }
                else if (caller.isDN && child.isDN)
                        drill_kernel(child);
                        }
        }
} // end drill_kernel


/**
 **  drill
 **
*/
public void drill (String[] words, int num_words) {
int i, size;
Node node, tn;

// HANDLE IDENTIFYING THE CORRECT NODE TO START ON
        if (num_words==1)
                node = gui.lastNode;
        else
                node = find_node_named(words[1]);

// CHECK FOR ERRORS
        if (node==null) {
        if (num_words>1)
             GUI.WARNING(0,"DataSea.drill","Can't find node "+words[1]);
        else
             GUI.WARNING(0,"DataSea.drill","Neither Name given nor existing
lastNode.");
                return;
                }
        else
                GUI.P(0,"drill","Found node named '"+node.Name+"'");

// ------------------------------------------------------------
gui.show_node_once(node);
size = node.Links.size();

gui.global_str[0] = "Drilling down "+node.Name+" ";
drill_kernel(node);
gui.global_str[0] = gui.global_str[0] + ".";
gui.global_str_size = 1;

GUI.P(0,"drill", gui.global_str[0]);

} // end drill


/**
 **  test
 **
*/
public void test () {
int i, size;
Node tn;


gui.test();

gui.show_node_once(find_node_named("Bob"));
       gui.sleep(500);
gui.show_node_once(find_node_named("phone"));
       gui.sleep(500);
gui.show_node_once(find_node_named("Web"));
       gui.sleep(500);
gui.show_node_once(find_node_named("Directory"));
       gui.sleep(500);

} // end test




/**
 **  set_POV
 **
*/
public void set_POV () {
int i, size;
Node tn;

if (GUI.lastNode != null) {
       POV = GUI.lastNode;
       }

} // end set_POV


/**
 **  repeat_priorCommand
 **
*/
       public void repeat_priorCommand () {

if (gui.priorCommand != null) {
       GUI.P(0,"repeat_priorCommand", "repeating "+gui.priorCommand);
       gui.input.string_input(gui.priorCommand);
       }
return;
} // end repeat_priorCommand
```

```
/**
 **  reset_mags
 **
*/
public void reset_mags () {
        reset_mags(Node.DEFAULT_MAG);
        return;
}// end reset_mags

/**
 **  reset_mags
 **
*/
        public void reset_mags (double newmag) {
         int size, i, j, child_size;
         Node node, child;
        double max_CS=0;

size = node_vec.size();
gui.text_threshold = gui.DEFAULT_TEXT_THRESHOLD;
Root.set_mag(Node.MAX_MAG); // to keep others from being normalized to MAX_MAG
for (i=0; i<size; i++) {
        node = (Node)(node_vec.elementAt(i));
        max_CS = get_max_CS(node);
        node.set_mag(newmag*max_CS); // 12/24/99 added max_CS
        }
} // end reset_mags


/**
 **  reset_ANmags_no_history
 **
*/
        public void reset_ANmags_no_history (double newmag) {
         int size, i, j, child_size;
         Node node, child;
        double max_CS=0;

size = node_vec.size();
gui.text_threshold = gui.DEFAULT_TEXT_THRESHOLD;
Root.set_mag(Node.MAX_MAG); // to keep others from being normalized to MAX_MAG
for (i=0; i<size; i++) {
        node = (Node)(node_vec.elementAt(i));
        if (node.isAN)
                node.set_mag_no_history(newmag);
        }
} // end reset_ANmags_no_history
```

```
/**
 **  reset_CSs
 **
*/
        public void reset_CSs () {
         int size, j_size, i, j;
         Node node;
        Link siblink;

size = node_vec.size();
for (i=0; i<size; i++) {
        node = (Node)(node_vec.elementAt(i));
        node.min_mag = Node.MIN_MAG;
        j_size = node.Links.size();
        for (j=0; j<j_size; j++) {
                siblink = node.getLink(j);
                if (i<10)
                        GUI.P(0,"DataSea.reset_CSs","resetting CS to
"+Link.DEFAULT_CS+
                                " for node <"+node.Name+"> and
<"+siblink.NodeR.Name+">");
                if (siblink!=null)     {
                        siblink.set_CS(Link.DEFAULT_CS);
                        }
                }
        }
} // end reset_CSs

/**
 **  self_mag
 **
public void self_mag (Node node) {
int i, j, size, child_size;
Node child, grand_child;
double CS, max_CS;

size = node_vec.size();
for (i=0; i<size; i++) {
        child = (Node)(node_vec.elementAt(i));
        child_size = child.Links.size();
        max_CS = 0;
        for (j=0; j<child_size; j++) {
                grand_child = child.getNodeAtLink(j);
                CS = child.getCS(grand_child);
                if (CS > max_CS)
                        max_CS = CS;
                }
        for (j=0; j<child_size; j++) {
                grand_child = child.getNodeAtLink(j);
                if (CS > max_CS)
```

```
                        max_CS = CS;
                    }
                }

        } // end self_mag
        */


        /**
         **  recent
         **
        */
        public void recent () {
        int i, size;
        long min_TS, max_TS, median_TS;
        Node tn;
        Node bob, files;

        min_TS = GUI.current_TS;
        max_TS = 0;

        size = node_vec.size();
        /********************************
        for (i=0; i<size; i++) {
                tn = (Node)(node_vec.elementAt(i));
                if (tn.created_TS < min_TS)
                        min_TS = tn.created_TS;
                if (tn.created_TS > max_TS)
                        max_TS = tn.created_TS;
        } // now we have min and max

        median_TS = min_TS + (max_TS - min_TS)/2;
        ***********************************/


        for (i=0; i<size; i++) {
                tn = (Node)(node_vec.elementAt(i));
                if (tn.created_TS > GUI.current_TS-30000)
                        tn.set_mag(tn.mag * 2);
        }


        /************************************
        GUI.P(0,"recent","min="+min_TS+", max="+max_TS+", median="+median_TS);
        bob = find_node_named("Bob");
        if (bob != null)
                GUI.P(0,"recent","Bob.created_TS="+bob.created_TS);
        files = find_node_named("Files");
        if (files != null)
                GUI.P(0,"recent","Files.created_TS="+files.created_TS);
```

```
************************************/

} // end recent

/**
 **  word_fn      run a function on selected_node(s)
*/
        public void word_fn (String[] words, int num_words) {
         Node node, tnode;
        Node node_array[];
        int i, j, size;

             GUI.P(0,"DataSea.word_fn","Started.");

        node_array = new Node[2];

        node_array[0] =  (Node)(gui.selected_nodes_vec.elementAt(0));
        node_array[1] =  (Node)(gui.selected_nodes_vec.elementAt(1));
        if (node_array[0]==null)
                {
                GUI.ERROR(0,"word_fn","node_array[0] is null");
                return;
                }
        if (node_array[1]==null) {
                GUI.ERROR(0,"word_fn","node_array[1] is null");
                return;
                }

GUI.P(0,"fn:", node_array[0].Name+"                   "+node_array[1].Name);
GUI.P(0,"fn:", (node_array[0].getNodeAtLink(1)).Name+"
"+(node_array[1].getNodeAtLink(1)).Name);
GUI.P(0,"fn:", (node_array[0].getNodeAtLink(2)).Name+"
"+(node_array[1].getNodeAtLink(2)).Name);
GUI.P(0,"fn:", (node_array[0].getNodeAtLink(3)).Name+"
"+(node_array[1].getNodeAtLink(3)).Name);


/**********************************************
for (j=0; j<2; j++) {
        node = node_array[j];
        size = node.Links.size();
GUI.P(0,"word_fn",node.Name+" size is "+size);
        for (i=0; i<size; i++)
        {
                tnode = node.getNodeAtLink(i);
                if (tnode!=null)
                {
                        if (tnode.isAN)
                        {
```

```
                        GUI.P(0,"word_fn", "DN "+node.Name+" is connected to
"+tnode.Name);
                        }
                }
        }

}
*******************************************/
} // end word_fn



/**
 **  con    find a CN for lastnode, mag children of that CN
 **
*/
public void con () {
int i, size, ti, tsize=0;
Node tn, ttn=null;

if (gui.lastNode == null)
        return;

size = gui.lastNode.Links.size();
for (i=0; i<size; i++) {
        tn = (Node)(gui.lastNode.getNodeAtLink(i));
        if (tn.isCN) {
                tn.more_mag(gui.lastNode);
                // NEED A SIMPLER WAY TO SIMPLY MAGNIFY IMMEDIATE SURROUND
                tsize = tn.Links.size();
                for (ti=0; ti<tsize; ti++) {
                        ttn = (Node)(tn.getNodeAtLink(ti));
                        ttn.more_mag(tn);
                        }
                }
}

return;
} // end con


/**
 **  SS      create a 'spread-sheet', using DNs connected to 2 selected AN's
*/
        public void SS (String[] words, int num_words) {
        Node AN1=null, AN2=null, DN1=null, DN2=null, SS_node=null;
        int i;
        int line_num=1;

        gui.global_str_size=0; // Must reset if creating new global_str
```

```
          GUI.P(0,"DataSea.SS","Started.");

          if (num_words == 1) // no args given, use gui.lastNode
          {
          DN1 = gui.lastNode;
          }
          if (num_words == 2) // one args given, use it
          {
                  DN1 = find_node_named(words[1]);
                  // CHECK FOR ERRORS
                  if (DN1==null)
                          {
                          GUI.ERROR(0,"DataSea.SS","Can't find node "+words[1]);
                          return;
                          }
          }
  // NOW WE HAVE DN1 TO OPERATE FROM

          DN2 = DN1.getDN(0);
                  if (DN2==null)
                          {
                          GUI.ERROR(0,"DataSea.SS","Null DN2 from DN1="+DN1.Name);
                          return;
                          }
  // NOW WE HAVE DN2 TO OPERATE FROM

          AN1 =  DN1.getAN(0);
          AN2 =  DN2.getAN(0);
          if (AN1==null)
                  {
                  GUI.ERROR(0,"SS","AN1 is null");
                  return;
                  }
          if (AN2==null) {
                  GUI.ERROR(0,"SS","AN2 is null");
                  return;
                  }
          DN1=null;
          DN2=null;

  // NOW WE HAVE AN1 and AN2 TO OPERATE FROM
  SS_node = new Node("SSheet","Form","Spread Sheet node",100, 100, 120,140);

  // Run through AN's, get all DN's
  gui.global_str[gui.global_str_size++] = AN1.Name+"                "+AN2.Name;
  gui.global_str[gui.global_str_size++] = "---------              --------------
  ";
  i=0; // initialize counter to catch all DN's of the AN's we have established
          DN1=AN1.getDN(i);
```

```
        if (DN1 != null)
              DN2=DN1.getDN_connected_to_AN(AN2);
        else
              DN2=null;
        SS_node.link(DN1);
        SS_node.link(DN2);
        set_child_position(SS_node, DN1, 0.01, 0.1*(++line_num));
        set_child_position(SS_node, DN2, 0.51, 0.1*(line_num));


        while (DN2!=null && gui.global_str_size<9)
        {
        gui.global_str[gui.global_str_size++] = DN1.Name+"
"+DN2.Name;
        i++;
        DN1=AN1.getDN(i);
        if (DN1 != null)
              DN2=DN1.getDN_connected_to_AN(AN2);
        else
              DN2=null;
        if ((DN1 != null) && (DN2 != null)) {
              SS_node.link(DN1);
              SS_node.link(DN2);
              set_child_position(SS_node, DN1, 0.01, 0.1*(++line_num));
              set_child_position(SS_node, DN2, 0.51, 0.1*(line_num));
              }
        }
line_num ++;
SS_node.link(AN1);
SS_node.link(AN2);
set_child_position(SS_node, AN1, 0.01, 0.1*(++line_num));
set_child_position(SS_node, AN2, 0.51, 0.1*(line_num));

reset_and_zoom("SSheet");

GUI.P(0,"SS","Done.");
return;
} // end SS


/**
 **  word_mail      create a 'mail' application    DEFUNCT
*/
        public void word_mail (String[] words, int num_words) {     // DEFUNCT
        Node name_node=null, address_node=null, target_node=null;
        Node MailForm_node=null, To_node=null;


         GUI.P(0,"DataSea.word_mail","Started.");
        if (num_words==1) {
                target_node = gui.lastNode;
                }
        else if (num_words==2) {
                target_node = find_node_named(words[1]);
                }
        if (target_node == null) {
                GUI.ERROR(0,"DataSea.word_mail","No node selected nor given as
name node.");
                return;
                }

        set_dist_start(target_node); // NEED FOR PROPAGATING

// FIND A  NAME
GUI.P(0,"DataSea.word_mail","Looking for 'name': ");
name_node=gui.lastNode.getAN_named("name",4);
        if (name_node == null) {
                GUI.ERROR(0,"DataSea.word_mail","No node found for name node,
given target_node '"
                        +target_node.Name+"'.");
                return;
                }

// FIND AN ADDRESS
GUI.P(0,"DataSea.word_mail","Looking for 'address': ");
address_node=gui.lastNode.getAN_named("address",4);
        if (address_node == null) {
                GUI.ERROR(0,"DataSea.word_mail","No node found for address node,
given target_node '"
                        +target_node.Name+"'.");
                return;
                }

DataSea.needdistUpdate = true; // Reset all Node.dist's


GUI.P(0,"DataSea.word_mail","Creating mail form for target node
'"+target_node.Name+"'.");
MailForm_node = new Node("MailForm","Form","", 50 , 50, 50, 50);
To_node = new Node("To: ","DN","",-20,10,1,1);
name_node.setX("word_mail", 30, 10);
address_node.setX("word_mail", 30, 30);
MailForm_node.link(address_node);
MailForm_node.link(name_node);
MailForm_node.link(To_node);

} // end word_mail    DEFUNCT
```

```
/**
 **  word_focus     make node the POV
*/
        public void word_focus (String[] words, int num_words) {
        Node node;

             GUI.P(0,"DataSea.word_focus","Started.");
// HANDLE IDENTIFYING THE CORRECT NODE TO START ON
        if (num_words==1)
                node = gui.lastNode;
        else
                node = find_node_named(words[1]);

// CHECK FOR ERRORS
        if (node==null)
                {
                GUI.ERROR(0,"DataSea.word_focus","Can't find node "+words[1]);
                return;
                }
// NOW, DO WHAT'S ASKED OF US
        POV=node;

} // end word_focus



/**
 **  find_max  Select for us the node with greatest mag
 **
*/
public void find_max () {
int i, size;
Node node=null, saved_node=null;
double saved_mag=Node.MIN_MAG;

size = node_vec.size();
for (i=0; i<size; i++) {
    node = (Node)(node_vec.elementAt(i));
        if (node.mag > saved_mag)  {
                saved_node = node;
                saved_mag = node.mag;
                }
        }
// Now, select for us the node with greatest mag
gui.lastNode = saved_node;
if (saved_node != null)
        GUI.P(0,"find_max","max mag is "+saved_node.mag+" for "+saved_node.Name);
else
```

```
        GUI.P(0,"find_max","No saved_node found.");
return;
} // end find_max



/**
 **  pulse  From all AN's, add AN.mag to children.
 **         Used after drawDN was false and not positioned
 **
*/
public void pulse () {
int i, size, j, j_size;
Node node=null, child=null;

GUI.P(0,"DataSea.pulse","Started.");
GUI.P(0,"DataSea.pulse","Started.");

size = node_vec.size();
for (i=0; i<size; i++) {
        node = (Node)(node_vec.elementAt(i));
        j_size = node.Links.size();
        for (j=0; j<j_size; j++) {
                child = (Node)(node.getNodeAtLink(j));
                if (child.isDN) {
                child.set_mag(child.mag + node.mag);
                }
        }
}

GUI.P(0,"DataSea.pulse","Done.");


return;
} // end pulse




/**
 **  word_select      select node    'find'->set lastNode, 'select'->set
node.isSelected
*/
        public void word_select (String[] words, int num_words) {
        Node node=null;
        int i, size;

String string_without_command="";
if (num_words >= 2) {
        string_without_command = words[1];
        for (i=2; i< num_words; i++)
```

```
                string_without_command = string_without_command+" "+words[i];
        }

             GUI.P(1,"DataSea.word_select","Started.");
// HANDLE IDENTIFYING THE CORRECT NODE TO START ON
        if (num_words==1)
                node = gui.lastNode;
        else {
                if (words[0].equalsIgnoreCase("AN"))
                        node = find_node_named(words[1], "AN");
                else
                if (words[0].equalsIgnoreCase("DN"))
                        node = find_node_named(words[1], "DN");
                else
                        node = find_node_named(string_without_command);
                }

// CHECK FOR ERRORS
        if (node==null)
                {
                GUI.WARNING(0,"DataSea.word_select","Can't find
"+string_without_command);
                return;
                }
// NOW, DO WHAT'S ASKED OF US
        gui.lastNode = node;
        gui.show_node_once(node);

if (words[0].equalsIgnoreCase("select")) {
        node.isSelected = true;
        gui.selected_nodes_vec.addElement(node);
        }

} // end select



/**
 **  word_unselect      unselect all nodes
*/
public void word_unselect (String[] words, int num_words) {
Node node, child;
int i, size;
GUI.P(0,"DataSea.word_unselect","Unselecting all nodes.");

if (words[0].equalsIgnoreCase("unselect") ||  words[0].equalsIgnoreCase("un")) {
        if (num_words == 2) {
                node = find_node_named(words[1]);
                if (node != null) {
                        unselect(node);
                        }
                } else {
                        unselect((Node)null);
                        }
                } // end check of 'unselect' command
        return;
        } // end word_unselect


        /**
         **  unselect      unselect
        */
        public void unselect (Node node) {
        Node child;
        int i, size;
        GUI.P(0,"DataSea.unselect","Unselecting all nodes.");

        // Loop on all nodes, unselect them and remove from list
        size = node_vec.size();
        for (i=0; i<size; i++) {
                child = (Node)(node_vec.elementAt(i));
                child.isSelected = false;
                gui.selected_nodes_vec.removeElement(child);
                }
        return;
        } // end unselect


        /**
         **  word_delete      delete node  WARNING!  dangerous method
        */
                public void word_delete (String[] words, int num_words) {
                Node node;

                        GUI.P(0,"DataSea.word_delete","Started.");
        // HANDLE IDENTIFYING THE CORRECT NODE TO START ON
                if (num_words==1)
                        node = gui.lastNode;
                else
                        node = find_node_named(words[1]);

        // CHECK FOR ERRORS
                if (node==null)
                        {
                        GUI.WARNING(0,"DataSea.word_delete","Can't find node "+words[1]);
                        return;
                        }

        // NOW, DO WHAT'S ASKED OF US
                node.unlink_all();
```

```
}// end word_delete (delete node);



/**
 **  word_release
*/
        public void word_release (String[] words, int num_words) {
        Node node=null;
        String name="UNKNOWN";

        GUI.P(0,"DataSea.word_release","Started.");

        if (POV == null) {
                GUI.WARNING(0,"DataSea.word_release","Need a POV.");
                return;
                }

// HANDLE IDENTIFYING THE CORRECT NODE TO START ON
        if (num_words==1)
                node = gui.lastNode;
        else
        if (num_words==2) {
                node = find_node_named(words[1]);
                if (node != null)
                        name = node.Name;
                }

// CHECK FOR ERRORS
if (node==null)
        {
        GUI.WARNING(0,"DataSea.word_release","Can't find node <"+name+">");
        return;
        }
else
        name = node.Name;

GUI.P(0,"DataSea.word_release","About to release '"+node.Name+"' from POV");
// NOW, DO WHAT'S ASKED OF US
        POV.unlink(node);
        node.unlink(POV);
return;

} // end word_release


/**
 **  word_unlink
*/
```

```
        public void word_unlink (String[] words, int num_words) {
        Node node1=null, node2=null;

            GUI.P(0,"DataSea.word_unlink","Started.");

        if (num_words<2) {
            GUI.ERROR(0,"DataSea.word_unlink","Need at least one argument
(unlink[link_between] node1 node2)");
            return;
        }
        if (num_words>3) {
            GUI.ERROR(0,"DataSea.word_unlink","Need at most two arguments
(unlink[link_between] node1 node2)");
            return;
        }
// HANDLE IDENTIFYING THE CORRECT NODE TO START ON
        if (num_words==2)
                {
                if (POV != null)
                        node1 = POV;
                else
                        node1 = gui.lastNode;
                node2 = find_node_named(words[1]);
                }
        if (num_words==3)
                {
                node1 = find_node_named(words[1]);
                node2 = find_node_named(words[2]);
                }
// CHECK FOR ERRORS
        if ((node1==null) || (node2==null))
                {
            GUI.ERROR(0,"DataSea.word_unlink","Can't find either node1 or
node2");
                return;
                }

GUI.P(0,"DataSea.word_unlink","About to unlink '"+node1.Name+"' from
'"+node2.Name+"'");
// NOW, DO WHAT'S ASKED OF US
        node1.unlink_both(node2);

} // end word_unlink


/**
 **   word_link
*/
        public void word_link (String[] words, int num_words) {
        Node node1=null, node2=null;
```

```
                GUI.P(0,"DataSea.word_link","Started.");

/**************
        if (num_words<2) {
                GUI.ERROR(0,"DataSea.word_link","Need at least one argument
(link[link_between] node1 node2)");
                return;
        }
**************/
        if (num_words>3) {
                GUI.ERROR(0,"DataSea.word_link","Need at most two arguments
(link[link_between] node1 node2)");
                return;
        }
// HANDLE IDENTIFYING THE CORRECT NODE TO START ON
        if ((num_words==1) && (POV!=null))
                {
                node1 = POV;
                node2 = gui.lastNode;
                }
        if (num_words==2)
                {
                node1 = gui.lastNode;
                node2 = find_node_named(words[1]);
                }
        if (num_words==3)
                {
                node1 = find_node_named(words[1]);
                node2 = find_node_named(words[2]);
                }
// CHECK FOR ERRORS
        if ((node1==null) || (node2==null))
                {
                GUI.ERROR(0,"DataSea.word_link","Can't find either node1 (or
lastNode) or node2");
                return;
                }

GUI.P(0,"DataSea.word_link","About to link '"+node1.Name+"' to
'"+node2.Name+"'");
// NOW, DO WHAT'S ASKED OF US
        node1.link(node2);

} // end word_link


/**
/**
 **  word_most
```

```
*/
        public void word_most (String[] words, int num_words) {
            GUI.P(0,"DataSea.word_most","NO CODE YET.");
// NOW, DO WHAT'S ASKED OF US
        ;
} // end word_most


/**

/**
 **  word_rename
*/
        public void word_rename (String[] words, int num_words) {
        Node node=null;
        String new_name="DEFAULT_NEW_NAME";

            GUI.P(0,"DataSea.word_rename","Started.");
// HANDLE IDENTIFYING THE CORRECT NODE TO START ON
        if (num_words<2)
               {
               GUI.ERROR(0,"DataSea.word_rename","Need at least one argument for
new name.");
               return;
               }
        if (num_words>3)
               {
               GUI.ERROR(0,"DataSea.word_rename","Need at most two arguments:
node and new_name.");
               return;
               }
        if (num_words==2) {
               node = gui.lastNode;
               new_name = words[1];
               }
        else
        if (num_words==3) {
               node = find_node_named(words[1]);
               new_name = words[2];
               }
// CHECK FOR ERRORS
        if (node==null)
               {
            GUI.ERROR(0,"DataSea.word_rename","Can't find node "+words[1]);
               return;
               }

// NOW, DO WHAT'S ASKED OF US
        node.Name = new_name;
} // end word_rename
```

```
/**
 **  and
 **
*/
        public void and (String[] words, int num_words) {
        Node node_1, node_2;
        int i;

        node_1 = find_node_named(words[1]);
        if (node_1==null)
                {
                GUI.ERROR(0,"and","node_1 is null");
                return;
                }
        node_2 = find_node_named(words[2]);
        if (node_2==null)
                {
                GUI.ERROR(0,"and","node_2 is null");
                return;
                }
        GUI.P(0,"and","OK: node_1 is "+node_1.Name+", node_2 is "+node_2.Name);


back_r((Node)node_1, node_2, 0);

} // end and



/**
 **  print_print_upstream
 **
*/
public void print_upstream (Node caller, Node node) {
int i, size;
Node child;

if (node == null) {
        GUI.ERROR(0,"DataSea.print_upstream","NULL node given");
        return;
        }

if (caller == null)
        System.out.println("");
else
        System.out.print(" -> "+node.Name+"{"+node.dist+"}");

size = node.Links.size();
```

```
for (i=0; i<size; i++) {
        child = node.getNodeAtLink(i);
        if (node.goesUpstreamTo(child)) {
                if (caller == null)
                        System.out.println(node.Name+"{"+node.dist+"}");
                print_upstream(node, child);
                }
        else
                System.out.println("");
        }

} // end print_upstream

/**
 ** method Back     like back(node), but set_Tdist is run on all children of
node
*/
        public void Back (String[] words, int num_words) {
        int i, size;
         Node node, child, grand_child;

        if (num_words<1)
                return;
        if (POV==null)
                return;

// HANDLE IDENTIFYING THE CORRECT NODE TO START ON
        if (num_words==1)
                node = gui.lastNode;
        else
                node = find_node_named(words[1]);

        if (words[0].equalsIgnoreCase("whats")) {
                whats = true;
                }

// CHECK FOR ERRORS
        if (node==null) {
        if (num_words>1)
            GUI.WARNING(0,"DataSea.Back","Can't find node "+words[1]);
        else
            GUI.WARNING(0,"DataSea.Back","Neither Name given nor existing
lastNode.");
                return;
                }
        else
                GUI.P(1,"DataSea.Back","Found node named '"+node.Name+"'");

// ------ for children of POV child --------------------
        child = POV.getNodeAtLink(0);
```

```
size = child.Links.size();
for (i=0; i<size; i++) {
        grand_child = (Node)(child.getNodeAtLink(i));
        GUI.P(0,"DataSea.Back","Working on POV child named `"+child.Name
                +"' grand_child named `"+grand_child.Name+"'");
        set_Tdist_start(grand_child); // ORDER FROM ALL CHILDREN OF POV CHILD,
THEN RUN back_r()
        back_r((Node)null, node, 0);
}
// ----- end for children of POV child -----------------


}// end Back


/*
 *
 */
public Node figure_out_node (String caller_fn, String[] words, int num_words) {
Node node;

        if (num_words<1)
                return((Node)null);
// HANDLE IDENTIFYING THE CORRECT NODE TO START ON
        if (num_words==1)
                node = gui.lastNode;
        else
                node = find_node_named(words[1]);
        if (node==null) {
            GUI.WARNING(0, caller_fn,"Null node, quitting.");
                }
        else
                GUI.P(0, caller_fn,"Found node named '"+node.Name+"'");
return(node);
} // end figure_out_node


/**
 ** method back3
*/
        public void back3 (String[] words, int num_words) {
         Node node;

if (null == (node = figure_out_node("back3", words, num_words)))
        return;

set_Tdist_start(node); // NEED FOR SETTING VARIABLE 'TDIST'

System.err.println("");
back2(node);
```

```
System.err.println("");
return;
} // end back3

/**
 **  back2
 **
*/
public boolean back2 (Node node) {
int i, size;
boolean ret_val = false;
Node child=null;


size = node.Links.size();
for (i=0; i<size; i++) {
child = node.getNodeAtLink(i);

if (child.dist == POV.dist+1) {
        ret_val = true;
        child.more_mag();
        System.out.println("+++   back2; next to POV, "
                +"mag++ of "+child.Name);
        return(ret_val);
        }

if (child.dist < node.dist) { // we may do something
//System.out.println("+++   back2; at POV+1 point, mag++ node="+node.Name+",
child="+child.Name);
        //System.out.println("   back2;'"+node.Name+"'-> recursing on
"+child.Name);
        if (true==back2(child)) {
                ret_val = true;
                System.out.println("+++   back2; path-point, child<node, "
                        +"mag++ of "+child.Name);
                child.more_mag(); // else ret_val is false
        }
        }
}
//System.err.println(" returning("+ret_val+")<-
("+node.Name+",Tdist="+node.dist+") ");
return(ret_val);
//else if (child.Type==node.Type) {} // recurse if same type and not next to POV
} // end back2



/**
 ** method backt     amplify mag going backwards, calls back_r
*/
```

```
        public void backt (String[] words, int num_words) {
        int i,j,k, size, size1, size2;
         Node node, child1=null, child2=null, child3=null;

/****************************************
if (null == (node = figure_out_node("backt", words, num_words)))
        return;
****************************************/
if (null == (node=POV)) {
        System.out.println("backt(): need a POV.");
        return;
        }

System.out.println("backt(): starting on POV.");

// 'node' is POV
// ------ for children --------------------
size = node.Links.size();
for (i=0; i<size; i++) {
        child1 = (Node)(node.getNodeAtLink(i));
        System.out.println("child1 is "+child1.Name);
        size1 = child1.Links.size();
        for (j=0; j<size1; j++) {
                child2 = (Node)(child1.getNodeAtLink(j));
                if (child2 != node) {
                System.out.println("   child2 is "+child2.Name);
                //else
                //System.out.println(" (skipping) "+child2.Name);
                }
                size2 = child2.Links.size();
                for (k=0; k<size2; k++) {
                        child3 = (Node)(child2.getNodeAtLink(k));
                        if (child3 != child1) {
                        System.out.println("           child3 is "+child3.Name);
                        System.out.println("
        pol='"+child2.getPol(child3)+"', isPolarized="+child3.isPolarized);
                        //else
                        //System.out.println("         (skipping) "+child3.Name);
                        if (child3.isEvent)
                                mag_downhill_event(child3);
                        }
                        }
                }
        }

// ----- end for children ----------------
return;
} // end backt
```

```
/**
 **  mag_downhill_event
 **
*/
public void mag_downhill_event (Node node) {
int i, size;
Node child;

if (node.isEvent)
        node.more_mag();
size = node.Links.size();
for (i=0; i<size; i++) {
        child = node.getNodeAtLink(i);
        System.out.println("Is '"+child.Name+"' downstream from '"+node.Name+"'?
"+node.goesDownstreamTo(child));
        if (node.goesDownstreamTo(child))
                mag_downhill_event(child);
        }

} // end mag_downhill_event


/**
 **  mark_distally
 **
*/
public void mark_distally (Node node) {
int i, size;
Node child;

node.isMarked = true;
size = node.Links.size();
for (i=0; i<size; i++) {
        child = (Node)(node.getNodeAtLink(i));
        if (child.dist > node.dist)
                mark_distally(child);
        }
return;
} // end mark_distally


/**
 ** method showdist
*/
public void showdist (Node node, int Tdist) {
Node child;
```

```
int i, size;

// ------ for children --------------------
size = node.Links.size();
for (i=0; i<size; i++) {
        child = (Node)(node.getNodeAtLink(i));
        if (child.Tdist == Tdist) {
                gui.dump_node(0, false, child);
                gui.show_node_once(child); // experiment
                gui.sleep(300);
        }
        if (child.Tdist > node.Tdist) // recurse only if we go distal
                showdist(child, Tdist);
}
// ----- end for children -----------------

return;
} // end showdist

/**
 ** method showdist
*/
public void showdist (String[] words, int num_words) {
Node node=null;
int i, size, Tdist=0;

if (num_words == 1) {
        if (gui.lastNode != null)
             node = gui.lastNode;
        else
                GUI.WARNING(0,"showdist", "no name given, nor lastNode");
        }
else if (num_words > 1) {
        node = find_node_named(words[1]);
        // CHECK FOR ERRORS
        if (node==null)
                {
                GUI.ERROR(0,"DataSea.showdist","Can't find node "+words[1]);
                return;
                }
        }
if (num_words > 2) {
if (words[2].equals("1"))
        Tdist = 1;
else
if (words[2].equals("2"))
        Tdist = 2;
else
if (words[2].equals("3"))
        Tdist = 3;
else
if (words[2].equals("4"))
       Tdist = 4;
else
if (words[2].equals("5"))
       Tdist = 5;
}
gui.getToolkit().sync(); // wait for things to calm down

System.err.println("showdist, Tdist set to "+Tdist);
set_Tdist_start(node);
showdist(node, Tdist);
return;
} // end showdist

/**
 ** method showdist

public void showdist (Node node) {
int i;
       if (node == null)
              return;

       add_POV();
       POV.link(node);
       set_dist_start(POV); // need for use with set_Tdist_recursive recursion
logic
       POV.setLinksVRparmsTo(node);  // USE THE EXISTING VR PARMS IN THE NEW
LINK
       GUI.P(0,"DataSea.showdist","Linking node "+node.Name);
       mag(node, "both", "+", 2);
       gui.lastNode = node;

} // end showdist(Node node)
*/


/**
 ** method showCNs
*/
public void showCNs () {
int i, size;
Node node, tn=null, CNode=null;

if (gui.lastNode == null)
       return;
node = gui.lastNode;

GUI.P(0,"DataSea.showCNs","Called on node "+node.Name);
gui.drawCN = true;
```

```
size = node.Links.size();
for (i=0; i<size; i++) {
        tn = (Node)(node.getNodeAtLink(i));
        CNode =  node.getCNodeAtLink(i);
        if (CNode != null) {
               GUI.P(0,"DataSea.showCNs","Found CNode "+CNode.Name);
               CNode.set_mag(Node.MAX_MAG);
        }

}
return;
} // end showCNs


/**
 ** method dumpCNs
*/
public void dumpCNs () {
int i, size;
Node node, tn=null, CNode=null;


// ------- for all nodes --------
size = node_vec.size();
for (i=0; i<size; i++) {
        tn = (Node)(node_vec.elementAt(i));
        if (tn.isCN)
               gui.dump_node(0, true, tn);
        }
// ---- end for all nodes -------

return;
} // end dumpCNs



/**
 **  freeze
 **
*/
public void freeze () {
int i, size;
Node child;


size = node_vec.size();
for (i=0; i<size; i++) {
        child = (Node)(node_vec.elementAt(i));
        child.isFrozen = true;
        }

} // end freeze

/**
 **  unfreeze_all
 **
*/
public void unfreeze_all () {
int i, size;
Node child;


size = node_vec.size();
for (i=0; i<size; i++) {
        child = (Node)(node_vec.elementAt(i));
        child.isFrozen = false;
        }

} // end unfreeze_all

/**
 **  unfreeze    Top Level version
 **
*/
public void unfreeze () {
Node node;

node = gui.lastNode;
if (node==null)
        return;

unfreeze_r(node);
return;
} // end unfreeze


/**
 **  unfreeze_r    Recursive version
 **
*/
public void unfreeze_r (Node node) {
Node child;
int i, size;

node.isFrozen = false;

size = node.Links.size();
for (i=0; i<size; i++) {
        child = node.getNodeAtLink(i);
        if (child.dist > node.dist)
              unfreeze_r(child);
        }
return;
} // end unfreeze_r


/**
 ** method show
*/
public void show (Node node) {
int i;
        if (node == null)
              return;

        GUI.P(0,"DataSea.show","Called on node "+node.Name);
        add_POV();
        POV.link(node);
        //set_dist_start(POV); // need for use with set_Tdist_recursive recursion
logic
        POV.setLinksVRparmsTo(node);  // USE THE EXISTING VR PARMS IN THE NEW
LINK
        GUI.P(0,"DataSea.show","Linking node "+node.Name);
        node.set_mag(Node.EMPHASIZED_MAG);
        gui.lastNode = node;
        needdistUpdate = true;

} // end show(Node node)


/**
 ** method show
*/
public void show (String input_string) {
Node node;
int num_words, jnum_words;
String jwords[];


         StringTokenizer st = new StringTokenizer(input_string, ",");
         num_words = st.countTokens();
         String[] words = new String[num_words];

 System.err.println("num_words is "+num_words+", input_string is
<"+input_string+">");

for(int i = 0; i < num_words; i++) {
        words[i] = st.nextToken();
        System.err.println("broken into words: <"+words[i]+">");
        if (i>0) {
```

```
                    Node nodex;
                    nodex = find_node_named(words[i]);
                    if (nodex != null) {
                    System.err.println("back_r on "+nodex.Name+",
Tdist="+nodex.dist);
                    back_r((Node)null, nodex, 0);
                    normalize();
                    }
                    }
        StringTokenizer sst = new StringTokenizer(words[i], " ,.<>\"\t\r\n" );
        jnum_words = sst.countTokens();
        if (jnum_words > 0) {
        jwords = new String[jnum_words];
        for(int j = 0; j < jnum_words; j++) {
                jwords[j] = sst.nextToken();
                if (i==0 && j==1) {
                        show(find_node_named(jwords[j]));
                        }
                }
        }
}
return;
} // end show


        public void show2 (String[] words, int num_words) {
        Node node;

if (null == (node = figure_out_node("show2", words, num_words)))
        return;

           show (node);
} // end show2


/**
 ** method show    alternate arguments
*/
        public void show2 (String word) {
        Node node;

        String s[] = new String[2];
        s[0] = new String ("show");
        s[1] = new String (word);
        show2(s, 2);

} // end show2


 /**
 ** method word_save
*/
public void word_save () {
        gui.magscale = 1;
         absorb_POV(true);
        reset_mags();
         reset_selected();
        gui.globalMaxPressure = 1.0;
        local(Root);
        simplify_recursive((Node)null, Root);
         } // end word_save

 /**
 ** method word_reset
*/
public void word_reset () {
        gui.global_str_size = 0;
        gui.magscale = 1;
        unfreeze_all();
         absorb_POV(false);
        reset_mags();
         reset_selected();
        gui.globalMaxPressure = 1.0;
        local(Root);
        simplify_recursive((Node)null, Root);
         } // end word_reset
 /**
 ** method reset_selected
*/
        public void reset_selected () {
         int i, size;
        double x, y;
         Node tn;

        gui.global_str_size = 0;

         size = node_vec.size();
         GUI.P(2,"reset_selected","Resetting selected");
         for (i=0; i<size; i++) {
             tn = (Node)(node_vec.elementAt(i));
             if (tn != null) {
                 tn.isSelected = false;
                 tn.isMarked = false;
                if (tn.isAN) {
                        tn.x = tn.X;
                        tn.y = tn.Y;
                        } else {
                         x = 0;
                        y = 0;
                        tn.x = x;
```

```
                    tn.y = y;
                }
            }

        }
    gui.selected_nodes_vec = new Vector(10);
    } // end reset_selected


    /**
     ** method word_zoom
    */
            public void word_zoom (String[] words, int num_words) {

            if (num_words == 1) {
                    if (gui.lastNode != null)
                        zoom(gui.lastNode.Name);
                    else
                    GUI.WARNING(0,"word_zoom", "no name given, nor lastNode");
                    }
            else if (num_words > 1) {
                  zoom(words[1]);
            }
        } // end word_zoom

        public void zoom (String name) {
        Node node;
        double i, smooth_motion_frames=2, temp_XOffset=0, temp_YOffset=0;


               node = find_node_named(name);
            if (node != null) {
               temp_XOffset = (node.x-gui.WindowXOffset) /smooth_motion_frames;
               temp_YOffset = (node.y-gui.WindowYOffset) /smooth_motion_frames;
               for (i=0; i<smooth_motion_frames; i++) {
                   gui.WindowXOffset +=(int)temp_XOffset;
                   gui.WindowYOffset +=(int)temp_YOffset;
                   gui.update(1);
                   }
                   GUI.P(1,"DataSea.zoom","Centering on
"+node.Name+",(x,y)=("+node.x+","+node.y+")");
                   GUI.P(1,"DataSea.zoom","WindowXOffset="+gui.WindowXOffset);
               }
            else
                {
                GUI.WARNING(0,"DataSea.zoom","Can't find node "+name+".");
            }
        } // end zoom
```

```
 /**
 ** method word_whats
*/
public void whats (String[] words, int num_words) {
String word1=null, word2=null, AnswerStr;
Node node1, node2, child;
int index, size, i;

        if (num_words<3) {
                GUI.WARNING(0,"whats", "Need three words, returning.");
                return;
                }


if (0 < (index = words[1].indexOf("'s"))) {
        word1 = words[1].substring(0,index); //  whats BOB'S hair
        node1 = find_node_named(word1);
        }
else {
        GUI.WARNING(0,"whats", "No possessive");
        return;
        }

word2 = words[2];
node2 = find_node_named(word2);
GUI.P(0,"whats", "Using word1="+word1+",
word2="+word2+"["+words[2]+"]["+node2.Name+"]");
if (node1==null) {
        GUI.WARNING(0,"whats", "node1 is null, returning.");
        return;
        }
if (node2==null) {
        GUI.WARNING(0,"whats", "node2 is null, returning.");
        return;
        }

// So, now we have two valid nodes. Find node between them.

size = node1.Links.size();
for (i=0; i<size; i++) {
        child = (Node)(node1.getNodeAtLink(i));
        if (child == node2.getParent()) {
                GUI.P(0,"whats", AnswerStr="What's "+words[1]+" "+words[2]+":");
                gui.global_str[gui.global_str_size=0] = AnswerStr;
                AnswerStr=" ANSWER: --> "+child.Name+" is "+words[1]+"
"+words[2]+":";
                GUI.P(0,"whats", AnswerStr);
                gui.global_str[gui.global_str_size=1] = AnswerStr;
```

```
               gui.global_str[gui.global_str_size=2] =
"=============================";
               }
        }
} // end whats

/**
 **  store_string_into_global_str
 **
*/
public void store_string_into_global_str (String str) {

gui.global_str[gui.global_str_size++] = str;

} // end store_string_into_global_str

 /**
 ** method word_trace
*/
public void word_trace (String[] words, int num_words) {
Node node1, node2;
Node child;
int i;


if (num_words==1)
       node1 = gui.SavedNode;
else
       node1 = find_node_named(words[1]);
if (node1 == null) {
        GUI.WARNING(0,"word_trace", "No node given or saved for node1");
       return;
        }

node2 = gui.lastNode;
if (node2==null) {
        GUI.WARNING(0,"word_trace", "Need a lastNode");
       return;
        }


GUI.P(0,"word_trace", "Tracing from <"+node1.Name+"> to <"+node2.Name+">");

boolean Saved_StopSpread = node1.StopSpread; // Temporarily change it
node1.StopSpread = true;

store_string_into_global_str("Explaining why <"+node2.Name+"> is enhanced:");
set_dist_start(node2); // set dist from node2, trace back from node1
trace(node1, node2);
set_dist_start(POV); // restore dist's
```

```
node1.StopSpread = Saved_StopSpread; // Restore it

return;
} // end word_trace

/**
 **  trace  call traceback on all children with path to POV
 **
*/
public void trace (Node node1, Node node2) {
int i, size;
Node child;
Link link=null;
String str="";

if (node1==null) {
        gui.WARNING(0,"trace", "node1 is null.");
        return;
        }
if (node2==null) {
        gui.WARNING(0,"trace", "node2 is null.");
        return;
        }

// node1.set_mag(Node.MAX_MAG);
// node1.isSelected = true;

// ------ for children --------------------
size = node1.Links.size();
for (i=0; i<size; i++) {
        child = (Node)(node1.getNodeAtLink(i));
        if (child.dist != -1) {
                if (child.hasSmallerDistThan(node1) && child.mag>=Node.MED_MAG) {
                link = node1.getLink(i);
                gui.p(str="trace: node ("+node1.Name+") has link named
["+link.Name+"] ... ");
                store_string_into_global_str(str);
                gui.p(str="trace:         ... to node ("+child.Name+")");
                store_string_into_global_str(str);
                trace(child, node2);
                }
                }
        }
// ----- end for children ----------------


} // end trace

/*
```

```
**
**/
public void traceback (Node node1, Node node2) { // mag and select paths back to
POV
Node parent;
String str;
int i, size;
Node child;
Link link=null;

// store_string_into_global_str(str);
//gui.global_str[gui.global_str_size++] = "Working on Node="+node1.Name;

if (node1 == null)
        return;
node1.set_mag(Node.EMPHASIZED_MAG);

// ------ for children --------------------
size = node1.Links.size();
for (i=0; i<size; i++) {
        child = (Node)(node1.getNodeAtLink(i));
        if (child.dist != -1) { // That is, there is a path to POV from here ...
                if (child.dist < (node1.dist-0.0001)) {
                link = node1.getLink(i);
                System.out.println("        <"+node1.Name+">("+node1.dist+")  ---
->"+link.Name+"---> <"+child.Name+">("+child.dist+") ");
                traceback(child, node2);
                }
                }
        }

return;
} // end traceback


/**
 ** method parse_set_cmd   set's the mag of a node
*/
public double parse_set_cmd (String[] words, int num_words) {
double val = 0;

if (gui.lastNode == null) {
        GUI.P(0,"parse_set_cmd",
        "No lastNode, value gotten is "+GUI.java_lang_double.valueOf(words[2]));
}

if (num_words != 3) {
        GUI.P(0,"parse_set_cmd","num_words wrong, = "+num_words);
        return(val);
        }
```

```
else {
        val = (double)(GUI.java_lang_double.valueOf(words[2])).doubleValue();
        if (words[1].equals("mag"))
                gui.lastNode.set_mag(val);
        if (words[1].equals("importance"))
                gui.lastNode.importance = val;
        }
// GUI.java_lang_double.valueOf(words[2]);
GUI.P(0,"parse_set_cmd", ""+ ( (double)1.0 +
(GUI.java_lang_double.valueOf(words[2])).doubleValue()));

return(val);
} // end parse_set_cmd



/**
 **  print_triplets   Print information based on triplet-configurations
 **
*/
public void print_triplets (Node node) {
int i, size, j, jsize;
Node child=null, grand_child=null;
String str; // to put a line of information into
String desc; // to get or set as Desc of link from AN to DN

gui.global_str_size = -1; // reset this

size = node.Links.size();
for (i=0; i<size; i++) {
        child = (Node)(node.getNodeAtLink(i));

        jsize = child.Links.size();
        for (j=0; j<jsize; j++) {
                grand_child = (Node)(child.getNodeAtLink(j));
                if (grand_child.isAN && child.isDN && child.mag >= Node.MED_MAG)
{
                        desc = grand_child.get_Desc(child);
                        if (desc.equals(""))
                                desc = "is";
                        str  = "("+gui.prec(child.mag, 3)+") "
                                +node.Name+"'s "+grand_child.Name+" "
                                +desc+" "+child.Name;
                        gui.global_str[++gui.global_str_size] = str;
                        GUI.P(0,"print_triplets",
gui.global_str[gui.global_str_size]);
                        }
                }
        }
```

```
return;
} // end print_triplets


/**
 **  print_blanks
 **
*/
public void print_blanks (int number_of_blanks) {
int i;

for (i=0; i<number_of_blanks; i++) {
        System.out.print(" ");
        }
return;
} // end print_blanks


/**
 **  print
 **
*/
public void print () {
int i, size;
Link link=null;
String s="";
Node child;

if (POV == null)
        return;


size = POV.Links.size();
for (i=0; i<size; i++) {
        child = POV.getNodeAtLink(i);
        if (child.isAN && !gui.drawAN)
                ;
        else {
                link = POV.getLink(i);
                s = "-->["+child.Name+" ("+child.mag+")("+link.Name+")]";
                s = s + "                      ";
                System.out.print( s.substring(0,20));
        print_r(POV, child, 1);
        }
        }
return;
} // end print
```

```
/**
 **  print_r
 **
*/
public void print_r (Node parent, Node child, int level) {
int i, size;
Node grand_child;
Link link=null;
int count=0;
String s;

if (parent == null)
       return;
if (child == null)
       return;

size = child.Links.size();
for (i=0; i<size; i++) {
grand_child = child.getNodeAtLink(i);
if ((grand_child != parent)
       && (grand_child.dist > child.dist)) {
       if (child.isAN && !gui.drawAN)
              ;
       else {
       if (grand_child.mag > Node.BIG_MAG)
              {
              if (++count > 1)
                     print_blanks(20*level);
              link = child.getLink(i);
              s = "  ["+grand_child.Name+"
("+grand_child.mag+")("+link.Name+")]";
              s = s + "                    ";
              if (child.mag > Node.BIG_MAG) {// See if the prior node is big
also
                     System.out.print( s.substring(0,20));
                     print_r(child, grand_child, level+1); // Recurse w/
increment
              }
              else {
                     System.out.print("....."+ s.substring(0,20));
                     print_r(child, grand_child, level); // Recurse w/o
increment
              }
              }
       }
       }
       }

if (count == 0)
```

```
        System.out.println(""); // Only carriage return when no valid children
exist

return;
} // end print_r



/**
 ** method print
*/
public void word_print (String[] words, int num_words) {
int i, size, max_dist=4;
Node node, tn, saved_node=null;
double saved_mag=0;


/********************************
node = gui.lastNode;
*******************************/

if (num_words==1)
        node = gui.lastNode;
else
       node = find_node_named(words[1]);
if (node == null) {
        GUI.WARNING(0,"DataSea.print", "No node to start on.");
        return;
        }
// So, continue ...


// PRINT TRIPLETS
print_triplets(node);

if (GlobalVec == null)
        GlobalVec = new Vector();
else
        GlobalVec.removeAllElements();

store_ANs_r(node, max_dist, 0); // this puts all the ANs within dist=max_dist
into the GlobalVec

size = GlobalVec.size();
for (i=0; i<size; i++) {
    tn = (Node)(GlobalVec.elementAt(i));
        if (tn.mag > saved_mag) {
                if (saved_node != null)
                saved_mag = tn.mag;
                saved_node = tn;
```

```
                    }
               }
        // now, saved_mag and _node have the biggest AN within max_dist
        if (saved_node==null) {
               GUI.WARNING(0, "word_print","Didn't process mag's well, saved_node ==
        null");
               return;
               }
        else
               GUI.P(0, "words_input","Max AN mag of "+saved_node.Name+" is
        "+saved_node.mag);

        // gui.global_str_size = -1;   used by print_triplets also
        System.err.println(" ============================================== ");
        System.err.println("Predominant Categories from "+saved_node.Name);
        gui.global_str[++gui.global_str_size] = "Predominant Categories from
        "+saved_node.Name;
        size = saved_node.Links.size();
        for (i=0; i<size; i++) {
               tn = (Node)(saved_node.getNodeAtLink(i));
               if (tn.isAN && tn.dist <= saved_node.dist && tn.dist > gui.lastNode.dist)
        {
                      System.err.print("---------------------- "+saved_node.Name);
                      gui.global_str[++gui.global_str_size] = "-----------------------
        ---- "+saved_node.Name;
                      go_backwards_r(tn);
                      System.err.println("");
                      }
               }


        if (POV != null) {
        // Look at high-mag children and grandchildren of current POV
        // ------ for children of POV --------------------
        int i_size, j_size, j;
        Node child=null, grand_child=null;

        gui.global_str[++gui.global_str_size] = "POV Children and Grandchildren:";

        i_size = POV.Links.size();
        for (i=0; i<i_size; i++) {
               child = (Node)(POV.getNodeAtLink(i));
               if (child.mag >= Node.BIG_MAG)
                      gui.global_str[++gui.global_str_size] = "
        "+child.Name+"("+child.mag+")";
               j_size = child.Links.size();
               for (j=0; j<j_size; j++) {
                      grand_child = (Node)(child.getNodeAtLink(j));
                      if (grand_child.mag >= Node.BIG_MAG)
```

```
                        gui.global_str[++gui.global_str_size] = "
"+grand_child.Name+"("+grand_child.mag+")";
                }
}
// ----- end for children of POV ----------------
}

System.out.println("TEST.....print_upstream().....................");
print_upstream((Node)null, node);
System.out.println("TEST.....print_upstream() done................");

System.err.println(" ================================================= ");
return;
} // end print



/**
 **  go_backwards
 **
*/
public void go_backwards_r (Node node) {
int i, size;
Node tn;

System.err.print(" -> "+node.Name);
gui.global_str[gui.global_str_size] = gui.global_str[gui.global_str_size]+"  ->
"+node.Name;

size = node.Links.size();
for (i=0; i<size; i++) {
        tn = (Node)(node.getNodeAtLink(i));
        if (tn.isAN && tn.dist < node.dist && tn.dist > gui.lastNode.dist)
                go_backwards_r(tn);
        }
return;
} // end go_backwards_r

/**
 **  store_ANs
 **
*/
public void store_ANs_r (Node node, int max_dist, int this_dist) {
int i, j, size, tsize;
Node tn;
boolean not_found=true;

this_dist++;
size = node.Links.size();
for (i=0; i<size; i++) {
        tn = (Node)(node.getNodeAtLink(i));
        not_found = true;
        if (tn.isAN) {
                if (GlobalVec.contains(tn))
                                not_found = false;
/*****************************************************************************
*****           tsize = GlobalVec.size(); // See if we have it already
*****           for (j=0; j<tsize; j++) {
*****                   if ((Node)(GlobalVec.elementAt(j)) == tn)
*****                           not_found = false;
*****                   }
*****************************************************************************/
                if (not_found) {
                        GlobalVec.addElement(tn);
                        }
                }
        if ((this_dist <= max_dist) && node.goesUpstreamTo(tn)) // recurse
                store_ANs_r(tn, max_dist, this_dist);
        }
} // end store_ANs


/**
 ** method word_dump
*/
        public void word_dump (String[] words, int num_words) {
         Node node;

if (num_words==0) {
        GUI.WARNING(0,"DataSea.word_dump","No words received at all.");
        return;
        }
if (num_words==1)
        GUI.P(1,"DataSea.word_dump","One word received '"+words[0]+"'.");
if (num_words==2)
        GUI.P(1,"DataSea.word_dump","Two words received '"+words[0]+"',
'"+words[1]+"'.");
if (num_words==3)
        GUI.P(1,"DataSea.word_dump","Three words received '"+words[0]+"',
'"+words[1]+"', '"+words[2]+"'.");

        if (num_words==1)
         node = gui.lastNode;
        else
         node = find_node_named(words[1]);
         if (node != null) {
                if (words[0].equals("d"))
                    gui.dump_node(0, false, node);
                else
                    gui.dump_node(0, true, node);
                }
        } // end word_dump

        /**
         ** method undo
        */
                public void undo (int levels) {
                 int i, size;
                 Node tnode;

                GUI.P(1,"undo","Begun, "+levels+" levels.");
                size = node_vec.size();
                for (i=0; i<size; i++)
                        {
                        tnode = (Node)(node_vec.elementAt(i));
                        tnode.undo(levels);
                        }
        } //end undo


        /**
         ** method word_less
        * ************************************************************
                public void word_less (String[] words, int num_words, int max_dist) {
                 Node tn=null;

                if (num_words==1)
                 tn = gui.lastNode;
                else
                 tn = find_node_named(words[1]);
                 if (tn != null) {
                    mag(tn, "both", "-");
                    }
                 else
                    GUI.WARNING(0,"DataSea.word_less","Can't find node "+words[1]+".");
                 } // end word_less
        *************************************************************/



        /**
         **  parse_event_input
         **
        */
                public double parse_event_input(String s) {
                 double offset = 0;

                offset = 0;
                if (s.equalsIgnoreCase("tomorrow"))
```

```
                    offset =  .01;  // was .66
               if (s.equalsIgnoreCase("today"))
                    offset =  .01;  // was .33
               if (s.equalsIgnoreCase("yesterday"))
                    offset =  .01;
               if (s.equalsIgnoreCase("noon"))
                    offset =  (1.0/24.0)*12.0 * 0.33;
               if (s.equalsIgnoreCase("1pm"))
                    offset =  (1.0/24.0)*13.0 * 0.33;
               if (s.equalsIgnoreCase("2pm"))
                    offset =  (1.0/24.0)*14.0 * 0.33;
               if (s.equalsIgnoreCase("3pm"))
                    offset =  (1.0/24.0)*15.0 * 0.33;
               if (s.equalsIgnoreCase("4pm"))
                    offset =  (1.0/24.0)*16.0 * 0.33;
        GUI.P(1,"parse_event_input","returning "+offset+" for string "+s);
        return(offset);

        } // end parse_event_input

        /**
         **  create_event  link caller to Event named Name with offset, return Event
         **
        */
               public Node create_event (Node caller, double offset, String Name) {
                int i;
               double yoffset=0;;
                Node event_node;

        if (caller==null) {
                GUI.ERROR(0,"DataSea.create_event","caller is null");
               return((Node)null);
               }

               Name = ""+event_counter++;

               // yoffset cycles up and down to space out overlapping events
               yoffset = 40.0 + (double)(((1000*offset) % 111)/4);
               GUI.P(1,"create_event","Creating event "+Name+", offset="
                       +offset+", positionX= "
                       +(caller.size_X*offset)+", yoffset="+yoffset);
               event_node = new Node(Name,"Event","",(int)(caller.size_X*offset),
                       (int)yoffset, 10,2);

               event_node.link(pop.TimeLine);
               return(event_node);
        } // end create_event
```

```
/**
 **  TS    TimeStamp a node: create event node, link it to arg and "Created"
 **
*/
public void TS (Node node) {
int i, size;
Node ts, created;

if (node == null)
       return;

GUI.P(0,"TS","TimeStamping "+node.Name);

if (null == (created = find_node_named("Created", "AN"))) {
       GUI.P(0,"TS"," Creating node 'Created'");
       created = pop.create_node("Created", "AN");
       created.link(Root);
       }
node = pop.create_node(""+GUI.current_TS, "Event"); // creates the event
node.link(created);           // link to "Created"
node.link(pop.TimeLine);      // link to TimeLine

return;
} // end TS


/**
 **  group   assumes the mag of distal ANs has been additively increased to
 **          emphasize ANs of more relevance to distal DNs.
 **          Only group at the level 'target_level'.
*/
       public void group (Node node, int target_level) {
        int i, size;
        Node tnode;

if (node == POV) {
gui.pressure_mag /= 3;
}
       if (node == null)
              {
              GUI.ERROR(0,"group","NULL node");
              return;
              }

       GUI.P(0,"group","level "+target_level);

       size = node.Links.size();

       if (node.dist < target_level) // we aren't there yet
       for (i=0; i<size; i++)
```

```
                {
                tnode = node.getNodeAtLink(i);
                if (tnode.dist > node.dist) // was >=, got infinite recursion w/
2 shows
                        group(tnode, target_level); // recurse
                }
        else
        if (node.dist == target_level) // we are at the target_level
        for (i=0; i<size; i++)
                {
                tnode = node.getNodeAtLink(i);
                if (tnode.isAN)
                        group_onto_biggest_AN(tnode);
                }
if (node == POV) {
gui.pressure_mag *= 3;
}
        return;
} // end group

/**
 **  group_onto_biggest_AN   force node near its AN with largest 'mag'
 **
*/
        public void group_onto_biggest_AN (Node node) {
         int i, size;
         Node tnode, temp_AN=null;
        double temp_AN_mag=0;

        if (node == null)
                {
                GUI.ERROR(0,"group_onto_biggest_AN","NULL node");
                return;
                }
        size = node.Links.size();
        for (i=0; i<size; i++)
                {
                tnode = node.getNodeAtLink(i);
                if (tnode.isAN && tnode.mag > temp_AN_mag) {
                        temp_AN = tnode;
                        temp_AN_mag = tnode.mag;
                        }
                }
// Presumably we now have the largest connected AN to given node

        if (temp_AN != null) {  // force near to AN
                node.x = (gui.random()-0.5) + temp_AN.x;
                node.y = (gui.random()-0.5) + temp_AN.y;
                node.z = (gui.random()-0.5) + temp_AN.z;
                }
```

```
} // end group_onto_biggest_AN

/**
 **  link_DN_to_ANs
 **
*      public void link_DN_to_ANs(Node dn_node, Node an_node) {
*         int i, size;
*         Node tnode;
*
*      if (an_node == null)
*              {
*              size = dn_node.Links.size();
*              GUI.P(0,"link_DN_to_ANs","   NULL an_node (start), size of
dn_node is "+size);
*              for (i=0; i<size; i++)
*                      {
*                      tnode = dn_node.getNodeAtLink(i);
*                      link_DN_to_ANs(dn_node, tnode);
*                      return;
*                      }
*              }
*      else if (!an_node.isAN)
*              {
*              GUI.P(0,"link_DN_to_ANs","                     an_node is not
Type AN");
*              return;
*              }
*      else
*              {
*              dn_node.link(an_node); // link() makes sure its not redundant
*              size = an_node.Links.size();
*              GUI.P(0,"link_DN_to_ANs","   recursing, linked "+an_node.Name+"
to "
*                      +dn_node.Name+", size of "+an_node.Name+" is "+size);
*              for (i=0; i<size; i++)
*                      {
*                      tnode = an_node.getNodeAtLink(i);
*                      link_DN_to_ANs(an_node, tnode);
*                      return;
*                      }
*              }
*
*
* } // end link_DN_to_ANs
*/


/**
 **  app_email
 **
*/
       public void app_email () {
        int i;
        Node app_node, to_string_node, from_string_node, to_node, from_node,
emailform_text_node;
       to_node = getNodeInNeighborhood("email");
       from_node = pop.create_node("rocky@tallis.com", "DN","email address for
Rocky Nevin");
       if (to_node == null) {
              GUI.WARNING(0,"app_email","getNodeInNeighborhood didn't return a
good node");
              return;
              }

app_node = new Node("EMF","Form","EMail Form",100, 100, 40, 80);
emailform_text_node = new Node("text","DN","EMail text",0, 0, 5, 1);
to_string_node = new Node("To:","DN","",0, 0, 5,1);
from_string_node = new Node("From:","DN","",0, 0, 5,1);
Root.link(app_node);
app_node.link(to_string_node);
app_node.link(from_string_node);
app_node.link(emailform_text_node);
app_node.link(to_node);
app_node.link(from_node);
// to_string_node.link(to_node);
// from_string_node.link(from_node);
// emailform_text_node.link(app_node);  done above

// the Y offset starts at the bottom and goes up, as y does in data space
set_child_position(app_node, to_string_node, 0.01, 0.8);
set_child_position(app_node, from_string_node, 0.01, 0.9);
set_child_position(app_node, to_node,  0.4, 0.8);
set_child_position(app_node, from_node,  0.4, 0.9);
// set_child_position(to_string_node, to_node, 1.4, 1);
// set_child_position(from_string_node, from_node, 2.0, 1);
set_child_position(app_node, emailform_text_node, 0.01, 0.5);

GUI.P(1,"app_email","Done, select EmailForm now, in VR mode");

needdistUpdate = true;

reset_and_zoom("EMF");
return;
} // end app_email


/**
 **  reset_and_zoom
 **
```

```
*/
        public void reset_and_zoom (String target_name) {
        Node target_node;

target_node = find_node_named(target_name);
if (target_node == null) {
        GUI.WARNING(0,"reset_and_zoom","can't find target node named
"+target_name);
        return;
        }
needdistUpdate = true;
absorb_POV(false); // This stops the thread via gui.StopThreadRequest
gui.mode_obj.set_render_mode("VR");
mag(target_node, "both", "+");

return;
} // end reset_and_zoom

/**
 **  getNodeInNeighborhood
 **
*/
        public Node getNodeInNeighborhood (String target_name) {
        Node ret_node = null, tn=null, target_node=null;
        int size = 0, i;

        GUI.P(0,"getNodeInNeighborhood","Looking for "+target_name);

// CHECK REFERENCE POINT
        if (gui.lastNode == null) {
               GUI.WARNING(1,"getNodeInNeighborhood","gui.lastNode is null");
               return(ret_node);
               }

        target_node = find_node_named(target_name);

// CHECK TARGET POINT
        if (target_node == null) {
               GUI.WARNING(0,"getNodeInNeighborhood","target_node is null");
               return((Node)null);
               }

        set_Tdist_start(gui.lastNode); // NEED FOR SETTING VARIABLE 'TDIST'
        size = target_node.Links.size();
        for (i=0; i<size; i++) {
               tn = target_node.getNodeAtLink(i);
               GUI.P(1,"getNodeInNeighborhood", " i is "+i+", tn="+tn.Name+",
Type="+tn.Type+", Tdist="+tn.dist);
               if ((tn.isAN) && (tn.dist==target_node.dist-1)) {
                      ret_node = tn; // Got it
```

```
                        GUI.P(0,"getNodeInNeighborhood"," Got it:
"+ret_node.Name);
                }
        }

return(ret_node);

} // end getNodeInNeighborhood




/**
 **  set_child_position
 **
*/
        public void set_child_position (Node parent, Node child, int offsetX, int
offsetY) {
        double theta, delta_x, delta_y;
        Link link;

if (parent == null) {
        GUI.ERROR(0, "set_child_position","parent is null");
        return;
        }
if (child == null) {
        GUI.ERROR(0, "set_child_position","child is null");
        return;
        }

child.setX("set_child_position, parent="+parent.Name,
        offsetX,
        offsetY);

link = parent.getLinkTo(child);
link.setLinksVRparms(child);

} // end set_child_position


/**
 **  set_child_position
 **
*/
        public void set_child_position (Node parent, Node child, double
fraction_width, double fraction_height) {
        double theta, delta_x, delta_y;
        Link link;
```

```
if (parent == null) {
        GUI.ERROR(0, "set_child_position","parent is null");
        return;
        }
if (child == null) {
        GUI.ERROR(0, "set_child_position","child is null");
        return;
        }

//      child.setX("set_child_position, parent="+parent.Name,
//              (parent.size_X * fraction_width),
//              (parent.size_Y * (fraction_height-1)));

        delta_x = fraction_width - 0.5;
         delta_y = fraction_height - 0.5;

        theta = gui.get_angle(delta_x,delta_y);
        child.set_theta_offset(theta, "Called by set_child_position");
link = parent.getLinkTo(child);
link.setLinksVRparms(child);

} // end set_child_position

}       // End of class DataSea
```

```
import java.lang.*;

/*
 * This is Link.java     by Rocky Nevin
 * @version     0.4, 3/12/98
 *
 */

public class LinkObj extends Object {
String Name;
String Type;
String
Description;
double CS;
Node Node;
Node NodeL, NodeR;
LinkObj NextLink;
long Link_ID;

/**
 **    LinkObj   CONSTRUCTORS
 **
*/

        public LinkObj() {
         CS = 1;
//  RECURSES   this.Node = new Node();
        }

}       // End of class LinkObj
```

```
/**
 *
 * This is VRObj.java
 *
 * <pre>
 * </pre>
 * @version     0.1, 8/9/98 Begun, based on prior program, G.java for DataSea
 * @author      Rocky Nevin
 *
*/
import java.lang.*;

//
// VR object   holds info about the visual appearance on the screen,
//             as part of a Node object

public class VRObj extends Object {

double x, y, z;
double dx=5, dy=5, dz=5;  // Size of object
double px, py, pz;  // Pressure forces to move nodes
double theta;  // The angle from a horizontal line, left to right
double radius; // The radius of this obj's sphere of influence
String Appearance;

/**
 **   VRObj CONSTRUCTORS
 **
*/

        public VRObj () {
}



}       // End of class VRObj
```

```
/*
 * nsr_position_node
 */
public boolean nsr_position_node (Node parent, Node node) { //
boolean local_recurse=true;

if (parent==null) {
P(3,"nsr_position_recursive","NULL parent to node "+node.Name);
return(false);
}

P(3,"nsr_position_node","Starting on "+parent.Name+" -> "+node.Name);

if (node.Type.equals("FAB")) {
        local_recurse=nsr_position_FAB(parent, node);
        if (node.VR_node != VR_FAB_NODE)
                P(0,"nsr_position_node","WRONG VR_FAB connection");
        }
else if (node.Type.equals("MACH")) {
        local_recurse=nsr_position_MACH(parent, node);
        }
else  if (node.Type.equals("DIR")) {
        local_recurse=nsr_position_DIR(parent, node);
        }
else if (node.Type.equals("DIR_ENTRY")) {
        local_recurse=nsr_position_DIR_ENTRY(parent, node);
        }
else if (node.Type.equals("TIMELINE")) {
        local_recurse=nsr_position_TIMELINE(parent, node);
        }
else {
        node.x=parent.x + 50 + node.X;
        node.y=parent.y + 50*node.ChildNum + node.Y;
        node.z=parent.z;
        }
return(local_recurse);
} // end nsr_position_node


/**
 *   method nsr_position_FAB  a NSR function
 */
boolean nsr_position_FAB (Node caller, Node node) {
        node.x=caller.x+50;
        node.y=caller.y;
        node.z=caller.z;
P(3,"position_FAB","Starting on "+caller.Name+" -> "+node.Name);
return(true);
} // end position_FAB


/**
 *   method nsr_position_MACH  a NSR function
 */
boolean nsr_position_TIMELINE (Node caller, Node node) {

        node.x+=0.5*(caller.x+node.X-node.x);
        node.y+=0.5*(caller.y+node.Y-node.y);
//      node.z+=0.5*(caller.z+node.Z-node.z);
P(3,"nsr_position_TIMELINE","Starting on "+caller.Name+" -> "+node.Name);
return(true);
} // end nsr_position_TIMELINE

/**
 *   method nsr_position_MACH  a NSR function
 */
boolean nsr_position_MACH (Node caller, Node node) {

        node.x+=0.5*(caller.x+node.X-node.x);
        node.y+=0.5*(caller.y+node.Y-node.y);
//      node.z+=0.5*(caller.z+node.Z-node.z);
P(3,"nsr_position_MACH","Starting on "+caller.Name+" -> "+node.Name);
return(true);
} // end nsr_position_MACH


/**
 *   method nsr_position_DIR  a NSR function
 */
boolean nsr_position_DIR (Node caller, Node node) {
int i, size;
Node tnode;

P(3,"position_DIR","Starting on "+caller.Name+" -> "+node.Name);

        node.x=caller.x+50;
        node.y=caller.y;
        node.z=caller.z;

size = node.Links.size();
for (i=0; i<size; i++) { // check dist of all children, position if appropriate
        tnode = node.getNodeAtLink(i);
        if (tnode.dist == 1+node.dist)
                nsr_render_DIR_ENTRY(node, tnode);
        }

return(true);
} // end position_DIR
```

```
/**
 *   method nsr_position_DIR_ENTRY  a NSR function
 */
boolean nsr_position_DIR_ENTRY (Node caller, Node node) {
String Name;
int column_num=0;

Name=node.Name;
// CORRECT THIS PART
if (Name.equals("Name"))
        column_num=1;
else if (Name.equals("Phone"))
        column_num=2;
else if (Name.equals("Address"))
        column_num=3;

node.x=caller.x+50*column_num;
node.y=caller.y;
node.z=caller.z;

P(3,"position_DIR_ENTRY","Starting on "+caller.Name+" -> "+node.Name);

return(true);
} // end position_DIR_ENTRY


/**
 *   method nsr_render_FAB   a NSR function
 */
boolean nsr_render_FAB (Node caller, Node node) {
Point ChildPoint=new Point();
int width, height;

map(node.x, node.y, ChildPoint);
width=(int)(node.dx*magscale*node.mag);
height=(int)(node.dy*magscale*node.mag);
graphics.drawRect(ChildPoint.x, ChildPoint.y, width, height);
graphics.drawString("FAB", ChildPoint.x, ChildPoint.y);

P(3,"nsr_render_FAB",node.Name+"("+node.x+","+node.y+")");
return(true);
} // end nsr_render_FAB

/**
 *   method nsr_render_TIMELINE   a NSR function
 */
boolean nsr_render_TIMELINE (Node caller, Node node) {
Node tnode;
Point ChildPoint=new Point();
int width, height;

map(node.x, node.y, ChildPoint);
width=(int)(node.dx*magscale*node.mag);
height=(int)(node.dy*magscale*node.mag);
P(3,"nsr_render_TIMELINE",node.Name+"("+node.x+","+node.y+")");

graphics.drawRect(ChildPoint.x, ChildPoint.y, width, height);
graphics.drawString(node.Name, ChildPoint.x, ChildPoint.y);
return(true);
} // end render_TIMELINE


/**
 *   method nsr_render_MACH   a NSR function
 */
boolean nsr_render_MACH (Node caller, Node node) {
Node tnode;
Point ChildPoint=new Point();
int width, height;

map(node.x, node.y, ChildPoint);
width=(int)(node.dx*magscale*node.mag);
height=(int)(node.dy*magscale*node.mag);
P(3,"nsr_render_MACH",node.Name+"("+node.x+","+node.y+")");

graphics.drawRect(ChildPoint.x, ChildPoint.y, width, height);
graphics.drawString(node.Name, ChildPoint.x, ChildPoint.y);
return(true);
} // end render_MACH


/**
 *   method nsr_render_DIR   a NSR function
 */
boolean nsr_render_DIR (Node caller, Node node) {
Point ChildPoint=new Point();
int width, height;

map(node.x, node.y, ChildPoint);
width=(int)(node.dx*magscale*node.mag);
height=(int)(node.dy*magscale*node.mag);
graphics.drawRect(ChildPoint.x, ChildPoint.y, width, height);

P(3,"nsr_render_DIR",node.Name+"("+node.x+","+node.y+")");
return(true);
} // end nsr_render_DIR

/**
```

```
 *   method nsr_render_DIR_ENTRY   a NSR function
 */
boolean nsr_render_DIR_ENTRY (Node caller, Node node) {
Point ChildPoint=new Point();
int width, height;

map(node.x, node.y, ChildPoint);
width=(int)(node.dx*magscale*node.mag);
height=(int)(node.dy*magscale*node.mag);
graphics.drawRect(ChildPoint.x, ChildPoint.y, width, height);

P(3,"nsr_render_DIR_ENTRY",node.Name+"("+node.x+","+node.y+")");
return(true);
} // end nsr_render_DIR_ENTRY
```

WHAT IS CLAIMED IS:

1. A method for creating a highly connected network of nodes indicative of computer-readable data, including the steps of:

capturing data contained in at least one legacy database; and

structuring the captured data as a set of linked nodes, wherein each of the nodes includes at least one link to another one of the nodes, and the set of linked nodes is structured such that when one of the nodes is designated as a point of view, representations of the nodes can be displayed as a sea of node representations.

2. The method of claim 1, also including the steps of:

designating one of the nodes as the point of view; and

displaying said representations of the nodes as said sea of node representations, viewed from said point of view.

3. The method of claim 1, also including the step of implementing a query, said step of implementing the

query including the steps of:

(a) establishing a first point of view and displaying said representations of the nodes as a first sea of node representations whose point of view is said first point of view;

(b) invoking a command which determines the query; and

(c) in response to the command, displaying a changed sea of node representations which emphasizes information having greater relevance to the query and deemphasizes information having less relevance to the query.

4. The method of claim 3, wherein step (c) is performed in a manner that is tolerant to imprecision and errors in the query.

5. The method of claim 3, wherein the command specifies key words, and step (c) is performed in such a manner that the changed sea of node representations facilitates access to relevant data containing none of the key words.

6. The method of claim 3, wherein step (c) is

performed in such a manner that the changed sea of node representations facilitates finding of data that is similar to known data, without specifying characteristics of said data that is similar to known data.

7. The method of claim 3, wherein step (c) includes the step of displaying a smoothly changing sea of node representations which changes from the first sea of node representations to the changed sea of node representations with smooth changes in visual state, so as to provide information to a user regarding speed at which displayed node representations change and regarding which parameters of displayed node representations change.

8. The method of claim 2, wherein said sea of node representations includes virtual reality renderings.

9. The method of claim 1, wherein the nodes have identical structure but at least some of the nodes have different content.

10. A method for interactively exploring, accessing, and visualizing information in a highly connected

network of nodes, said method including the steps of:

determining a set of linked nodes, each of the nodes including at least one link to another one of the nodes, wherein the set of linked nodes is structured such that representations of the nodes can be displayed as a sea of node representations; and

designating one of the nodes as a point of view, linking a number of the nodes directly to the point of view, and calculating individual link distances from each of at least some of the nodes to the point of view, thereby determining a hierarchical network of the nodes which is amenable to visualization.

11. The method of claim 10, wherein there are cyclic loops in linkages between at least some of the nodes directly and the point of view.

12. The method of claim 10, also including the step of:

adding or deleting at least one link of at least one of the nodes, thereby changing the hierarchical network.

13. The method of claim 10, also including the step

of:

displaying representations of the nodes as a sea of node representations, viewed from said point of view.

14. The method of claim 10, wherein the hierarchical network of the nodes determines a connection strength of each of a set of linkages between at least some of the nodes, and a magnitude of each of at least some of the nodes, and wherein position and size of each of the nodes in said visualization is determined in accordance with each said connection strength and magnitude.

15. The method of claim 10, wherein said sea of node representations includes virtual reality renderings.

16. The method of claim 10, wherein each of the nodes has a node type, each of said link distances is determined by a function of the number of links between a pair of the nodes and the node type of each node of said pair, and the hierarchical network has a hierarchical tree structure.

17. The method of claim 10, also including the step

of:

implementing a user interface which displays representations of at least some of the nodes, wherein the user interface allows emulation of application programs by linking appropriate ones of the nodes.

18. The method of claim 10, also including the step of:

implementing a user interface which displays representations of at least some of the nodes, wherein the user interface implements a simple command and query syntax which is amenable to a voice interface.

19. A method, including the steps of:

structuring computer-readable data as a set of linked nodes, wherein each of the nodes includes at least one link to another one of the nodes, each of the nodes has a name associated therewith, and the set of linked nodes is structured such that when one of the nodes is designated as a point of view, representations of the nodes can be displayed as a sea of node representations; and

maintaining information specific to each of the nodes, including by maintaining the name of each of the nodes such that each said name is searchable and retrievable.

20. The method of claim 19, wherein the information specific to each of the nodes, includes a magnitude and connection strength of a link between said each of the nodes and at least one other one of the nodes.

21. A method for associating nodes of a set of linked nodes, wherein each of the nodes contains computer-readable data, at least one link to another one of the nodes, and a link identification for each event which links said each of the nodes to another one of the node, and wherein the set of linked nodes is structured such that when one of the nodes is designated as a point of view, representations of the nodes can be displayed as a sea of node representations, said method including the steps of:

storing, in an abstract node, a meaningful context common to a set of the nodes, wherein the abstract node is linked to each of the nodes in the set; and

sharing a single link identification among the nodes in said set, thereby associating the links that are identified by said single link identification.

22. The method of claim 21, also including the step of modulating a connection strength of the links that are identified by said single link identification, thereby sensitizing or desensitizing said links to further operations.

23. A method of establishing a set of linked nodes from text data, wherein each of the nodes includes at least one link to another one of the nodes, and the set of linked nodes is structured such that when one of the nodes is designated as a point of view, representations of the nodes can be displayed as a sea of node representations, said method including the steps of:

creating a full-text-node containing the text data;

discard selected words from the text data, thereby determining a set of remaining text, and creating a node for each word of the remaining text;

linking the full-text-node to each node representing one said word of the remaining text.

24. A method of establishing a set of linked nodes from data organized in rows and columns with column headings, wherein each of the nodes includes at least one link to another one of the nodes, and the set of linked nodes is structured such that when one of the

nodes is designated as a point of view, representations of the nodes can be displayed as a sea of node representations, said method including the steps of:

representing each of the column headings by an abstract node;

representing each cell of the data by a data node;

establishing links between each said abstract node and each said data node that corresponds to a cell in a column whose column heading is represented by said abstract node; and

establishing links between each said data node that corresponds to a cell in one of the rows.

25. A method of establishing a set of linked nodes from files linked by HTML references, wherein each of the nodes includes at least one link to another one of the nodes, and the set of linked nodes is structured such that when one of the nodes is designated as a point of view, representations of the nodes can be displayed as a sea of node representations, said method including the steps of:

establishing data nodes, each of the data nodes representing each of the files;

establishing links from said data nodes to terms found in the files.

26. The method of claim 25, wherein each of the terms is one of a set of selected tag values such as meta-tags or heading values.

27. The method of claim 25, also including the step of:

establishing links to abstract nodes representing suffixes of the files.

28. A method of establishing a set of linked nodes from files from computer file system, wherein each of the nodes includes at least one link to another one of the nodes, and the set of linked nodes is structured such that when one of the nodes is designated as a point of view, representations of the nodes can be displayed as a sea of node representations, said method including the steps of:

establishing links between data nodes representing a file directory and data nodes representing files or sub-directories in the file directory.

29. A method for retrieving data of interest from a network of linked nodes, wherein each of the nodes includes data and at least one link to another one of the nodes, and the set of linked nodes is structured such that when one of the nodes is designated as a point of view, representations of the nodes can be displayed as a sea of node representations, said method including the steps of:

(a) establishing a first point of view and displaying said representations of the nodes as a first sea of node representations whose point of view is said first point of view;

(b) invoking a command which determines a query; and

(c) in response to the command, displaying a changed sea of node representations which emphasizes information having greater relevance to the query and deemphasizes information having less relevance to the query.

30. The method of claim 29, wherein step (c) includes the step of:

tracing backwards from a target node to the first point of view by following all links from the target node to intermediate nodes having lesser magnitude, and displaying representations of said intermediate nodes.

31. The method of claim 29, wherein each of the node representations is displayed in a position that depends on parameter values of the corresponding node, and wherein step (c) includes the step of:

gradually changing the displayed position of at least one of the node representations, thereby showing a transition between an initial state and a final state of said one of the node representations.

32. The method of claim 31, wherein step (c) includes the step of:

operating on parameters indicative of pushing or pulling of the displayed position of said at least one of the node representations relative to displayed positions of others of the node representations.

33. The method of claim 29, wherein step (c) includes the step of:

traveling distally and upstream from a target node, finding the first abstract node linked to the target node and emphasizing a displayed representation of said first abstract node, thereby abstracting the target node.

34. The method of claim 33, wherein step (c) also includes the step of:

abstracting the target node at a higher level, by traveling from said first abstract node to directly linked abstract nodes which are both distal to and upstream of said first abstract node.

35. The method of claim 29, wherein step (c) includes the step of:

emphasizing displayed representations of abstract nodes linked to a target node which have not been recently visited by a query operation.

36. The method of claim 29, wherein step (c) includes the step of:

magnifying displayed representations of nodes based on similarity of each of the nodes to a chosen node, by magnifying displayed representations of data nodes linked to at least some of a set of abstract nodes linked to the chosen node.

37. The method of claim 29, wherein step (c) includes the step of:

modifying a potentiation parameter of at least one of the nodes.

38. A method of displaying node representations indicative of a network of linked nodes, wherein each of the nodes includes data and at least one link to another one of the nodes, and the set of linked nodes is structured such that when one of the nodes is designated as a point of view, representations of the nodes can be displayed as a sea of node representations, said method including the steps of:

designating one of the nodes as the point of view; and

displaying said representations of the nodes as said sea of node representations, viewed from said point of view, with visual emphasis assigned to each of the node representations dependent on parameters of each of the nodes, said parameters including connection strength of a link between said each of the nodes and at least one other one of the nodes.

39. The method of claim 38, wherein said parameters also include polarization of the link between said each of the nodes and at least one other one of the nodes.

40. The method of claim 38, wherein said parameters also include the minimum number of links between said each of the nodes and at least one other one of the nodes.

ABSTRACT OF THE DISCLOSURE

A computer implemented method of storing, manipulating, assessing, and displaying data and its relationships, and a computer system (with memory) programmed to implement such method. The data is stored into nodes, and visualized as a sea of linked nodes.